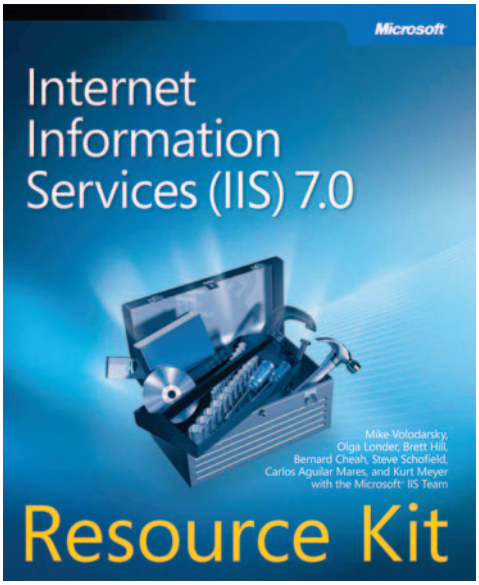


Internet Information Services (IIS) 7.0 Resource Kit

Mike Volodarsky, Olga Londer, Brett Hill, Bernard Cheah, Steve Schofield, Carlos Aguilar Mares, and Kurt Meyer with the Microsoft IIS Team

To learn more about this book, visit Microsoft Learning at http://www.microsoft.com/MSPress/books/9550.aspx

9780735624412

Microsoft Press

Table of Contents

Chapter 1

Introducing IIS 7.0

In this chapter:

Microsoft Internet Information Services (IIS) 7.0 in Windows Server 2008 is a Web server that provides a secure, easy-to-manage platform for developing and reliably hosting Web applications and services. IIS 7.0 has been completely redesigned and offers major advantages over previous versions of IIS. With its new modular and extensible architecture, IIS 7.0 makes developing, deploying, and configuring and managing Web applications and infrastructure easier and more efficient than ever before.

To put it simply, IIS 7.0 is the most powerful Microsoft Web server platform ever released. It provides an array of new capabilities that improve the way Web applications and services are developed, deployed, and managed. The modular design of IIS 7.0 gives administrators full control over their Web servers' functionality, providing an extensible architecture that enables administrators and developers to build customized and specialized Web servers. New administration capabilities and the distributed XML-based configuration system make deploying and managing Web applications on IIS 7.0 more straightforward and efficient than on any other Web server. In addition, new diagnostic and troubleshooting capabilities of IIS 7.0 enable administrators and developers alike to minimize potential downtime.

In this chapter, we will focus on the major new features and functionality in IIS 7.0 and their advantages over previous versions of IIS. We will also look at basic administration tasks and discuss the differences in the availability of IIS 7.0 features in Windows Server 2008 and Windows Vista.

Overview of IIS 7.0

IIS 7.0 provides features and functionality that enable administrators to reliably and effectively manage Web infrastructures; developers to rapidly build Web applications and services; and hosters to provide a cost-effective, scalable, and reliable Web hosting to a broad set of customers.

For administrators, IIS 7.0 provides a secure, reliable, and easy-to-manage Web server platform. The customizable installation of IIS 7.0 ensures that they can minimize the attack surface, patching requirements, and the memory footprint of their Web infrastructure. The IIS 7.0 process model makes Web sites and applications more secure by automatically isolating them, providing sandboxed configuration and unique process identity by default.

IIS 7.0 reduces management complexity, providing a set of tools that make administration of Web infrastructures more efficient. IIS Manager has a new task-based, feature-focused management console, which provides an intuitive user interface for administrative tasks. In addition to IIS Manager, there is also a new command line administration tool, a Windows Management Instrumentation (WMI) provider, and a .NET application programming interface (API).

IIS 7.0 supports simplified management of Web farms where Web server configuration can be stored together with Web application code and content on a centralized file server and can be shared across front-end Web servers on a farm.

IIS 7.0 enables administrators to securely delegate site and application administrative control to developers and content owners without administrative privileges on the server, thus reducing the administrative burden and cost of ownership. Using IIS Manager from Windows Vista, Windows XP, Windows Server 2003, or Windows Server 2008, developers and content owners can manage their sites and applications remotely while connected to a server over HTTPS from any location.

In addition, new troubleshooting and diagnostics capabilities in IIS 7.0 enable administrators to reduce Web server downtime.

For developers, IIS 7.0 provides a flexible, more extensible Web server platform for developing and deploying Web applications on Windows Server 2008 and Windows Vista. Developers can build applications on IIS 7.0 using the Web framework of their choice, including ASP.NET, classic ASP, PHP, PERL, ColdFusion, Ruby, and many others.

IIS 7.0 provides unprecedented extensibility. It has a fully componentized architecture, with more than 40 pluggable modules built on top of public extensibility APIs. Developers can create new or replacement modules in native or managed code, extend IIS configuration, and build IIS Manager extensions that plug in seamlessly to the management console.

IIS 7.0 has a distributed file-based configuration system that enables IIS settings to be stored in web.config files along with the ASP.NET settings. This unified configuration system simplifies development and enables applications to be xcopy-deployed, preconfigured, to IIS 7.0 servers.

In addition, new diagnostic capabilities, including access to run-time information and automatically tracing failed requests, help developers to troubleshoot issues quicker and minimize Web site downtime.

For hosters, IIS 7.0 provides a cost-effective, more scalable Web server platform for delivering reliable Web hosting to a broad set of customers. IIS 7.0 lowers costs by providing a new,

scalable shared hosting architecture that is capable of hosting thousands of Web sites on a single IIS 7.0 server without sacrificing isolation or reliability.

IIS 7.0 enables Web hosters to reach more customers by using a new FastCGI module that is capable of providing fast and reliable hosting for PHP and other Web frameworks.

In addition, IIS 7.0 provides a File Transfer Protocol (FTP) server that enables Web hosters to offer their customers a fully integrated Web/FTP platform with modern publishing capabilities, such as FTP over Secure Sockets Layer (SSL) and membership-based authentication.

What's New in IIS 7.0

IIS 7.0 has been completely redesigned and re-engineered from the ground up. The new features and functionality provide many new capabilities that enable administrators and developers to:

- Minimize patching and security risks with fine-grained control over the Web server footprint.
- Implement new Web solutions rapidly by using an extensibility framework.
- Go to market faster with simplified deployment and configuration of applications.
- Reduce administrative costs by managing Web infrastructures more efficiently.
- Reduce Web site downtime by quickly resolving faulty applications.

These advancements have been made possible because of major innovations in IIS 7.0, as follows:

- A modular, extensible core Web server
- A unified, distributed file-based configuration system
- Integrated health monitoring and diagnostics
- A set of new administration tools with delegation support

In addition, IIS 7.0 offers a new Windows Process Activation Service (WAS) that exposes IIS 7.0 processing model to both HTTP and non-HTTP based applications and services.

Let's look at these innovations and their advantages over previous versions of IIS in more detail.

Core Web Server

The IIS 7.0 core Web server has been completely redesigned and is very different from IIS 6.0. Its new, fully componentized architecture provides two fundamental enhancements that form a foundation for many advantages in security, performance, scalability, manageability, and flexibility. These two fundamental enhancements are modularity and extensibility.

Modularity

In previous versions of IIS, all functionality was built by default into a monolithic server. There was no easy way to extend or replace any of that functionality. In IIS 7.0, the core Web server has a completely modular architecture. All of the Web server features are now managed as stand-alone components. The IIS 7.0 Web core is divided into more than 40 separate components, each of which implements a particular feature or functionality. These components are referred to as modules. You can add, remove, and replace the modules depending on your needs.

In IIS 7.0, the ASP.NET run time is fully integrated with the core Web server, providing a unified request processing pipeline. Both native and managed code is processed through this single request pipeline. All notification events in the request pipeline are exposed to both native and managed modules. This integration enables existing ASP.NET features—including forms-based authentication, membership, session state, and many others—to be used for all types of content, providing a consistent experience across the entire Web application.

Figure 1-1 shows the unified request processing pipeline, with several stages shown at the beginning and at the end of request processing. At the Authenticate Request stage, Figure 1-1 shows authentication modules that are available for all requests. Basic Authentication, Windows Authentication, and Anonymous Authentication are native modules. Forms Authentication is a managed module. Both native and managed authentication modules provide services for any content type, including managed code, native code, and static files.

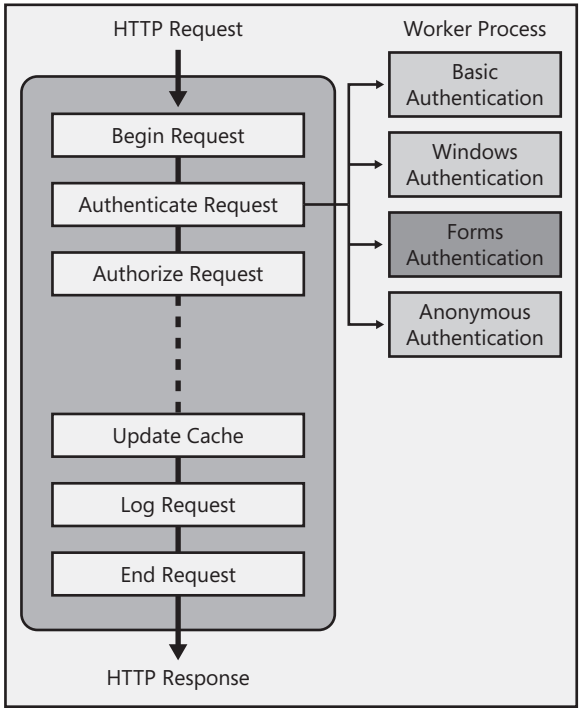


Figure 1-1 IIS 7.0 integrated request processing.

Note For more information on request processing, refer to Chapter 2, "Understanding IIS 7.0 Architecture."

IIS 7.0 modularity enables you to do the following:

- **Secure the server by reducing the attack surface area.** Reducing an attack surface area is one of the major steps to a secure system. In IIS 7.0, Web server features that are not required can be safely removed without affecting the functionality of your applications, thus reducing the attack surface area.
- **Improve performance and reduce memory footprint.** When you remove Web server features that are not required, the server's memory usage is reduced. In addition, the amount of code that executes on every request is reduced, leading to improved performance.
- **Build custom and specialized servers.** Selecting a particular set of server features and removing the ones that are not required allows you to build custom servers that are optimized for performing a specific function, such as edge caching or load balancing.

Note For more information on server modularity, refer to Chapter 3, "Understanding the Modular Foundation."

Extensibility

The modular architecture of IIS 7.0 enables you to build server components that extend or replace any existing functionality and add value to Web applications hosted on IIS.

The core Web server includes a new Win32 API for building core server modules. You can add custom features to extend or replace the existing Web server features with your own or third-party core Web server extensions built using this new extensibility API.

The core Web server modules are new and more powerful replacements for Internet Server Application Programming Interface (ISAPI) filters and extensions, although these filters and extensions are still supported in IIS 7.0. The new C++ extensibility model in IIS 7.0 uses a simplified object-oriented API that promotes writing robust server code to alleviate problems that previously plagued ISAPI development.

Moreover, IIS 7.0 also includes support for development of core Web server extensions using the .NET Framework. IIS 7.0 has integrated the existing IHttpModule API for ASP.NET, enabling custom managed code modules to access all events in the request pipeline, for all requests.

ASP.NET integration in IIS 7.0 enables server modules to be rapidly developed using capabilities of ASP.NET and the .NET Framework, instead of using the lower-level IIS C++ API. ASP.NET

managed modules are capable of fully extending the server and are able to service requests for all types of content including, for example, ASP, Common Gateway Interface (CGI), and static files.

Using ASP.NET or native C++ extensibility, developers can build solutions that add value for all application components, such as custom authentication schemes, monitoring and logging, security filtering, load balancing, content redirection, and state management.

Note For more information on core Web server extensibility, refer to Chapter 12, "Managing Web Server Modules."

Configuration

The early versions of IIS had few configuration settings, and they were stored in the registry. IIS 5.0 introduced a binary store called the metabase for managing URL-based configuration. In IIS 6.0, the binary metabase was replaced with an XML-based metabase to store configuration data. IIS 7.0 introduces a distributed XML file–based configuration system that enables administrators to specify settings for IIS and its features in clear text XML files that are stored with the code and content. The XML files hold the configuration settings for the entire Web server platform, including IIS, ASP.NET, and other components. The files store settings on the server, site, and application levels, and they may optionally be set at the content directories level together with the Web content, enabling delegated management.

Because Web site and application settings are no longer tied to a centralized configuration store on the local machine—as in previous versions of IIS—this distributed file-based configuration system dramatically simplifies application deployment by providing xcopy deployment of configuration together with application code and content. In addition, this configuration system enables sharing configuration for a site or application across a Web farm.

IIS 7.0 configuration is based on the .NET Framework configuration store. This common format enables IIS configuration settings to be stored alongside an ASP.NET configuration in a web.config files hierarchy, providing one configuration store for all Web platform configuration settings that are accessible via a common set of APIs and stored in a consistent format.

The distributed configuration hierarchy includes the global, computer-wide, .NET Framework configuration files, machine.config and root web.config, the global IIS configuration file applicationHost.config, and distributed web.config configuration files located within the Web sites, applications, and directories, as shown in Figure 1-2.

The .NET Framework global settings for a server machine are stored in the machine.config file located in the *%SystemRoot%*\Microsoft .NET\Framework \<version>\config folder. Global ASP.NET settings for a Web server are stored in the root web.config file located in the same folder on the server machine.

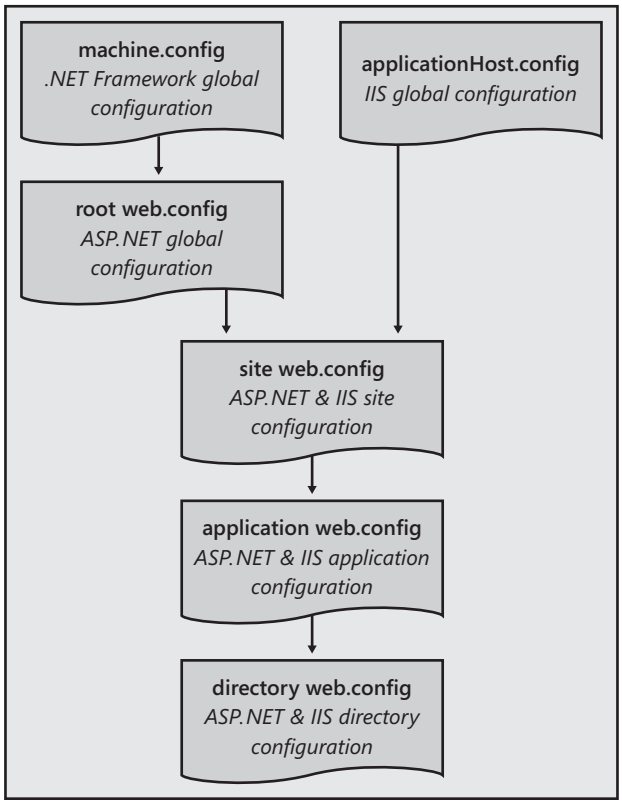


Figure 1-2 File-based distributed configuration store.

IIS 7.0 stores global configuration in the applicationHost.config file located in the *%SystemRoot%*\System32\Inetsrv\Config folder. ApplicationHost.config has two major configuration sections: <*system.applicationHost*> and <*system.webServer*>.

The <*system.applicationHost*> section contains settings for site, application, virtual directory, and application pools. The <*system.webServer*> section contains configuration for all other settings, including global Web defaults.

URL-specific configuration is stored in applicationHost.config via <location> tags. IIS 7.0 reads and writes URL-specific configuration in the web.config files hierarchy for sites, applications, and content directories on the server, along with ASP.NET configuration.

Figure 1-3 shows the structure of a site web.config file and its inheritance from global configuration files.

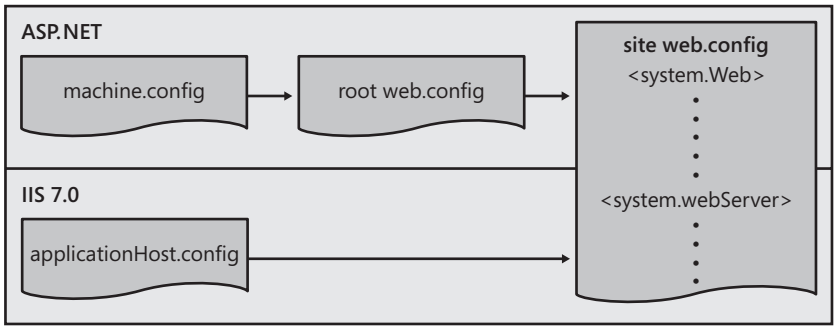


Figure 1-3 Site web.config file.

The server administrator may delegate different levels of the configuration hierarchy to other users, such as the site administrator or the application developer. By default, write access to configuration settings is limited to the server administrator only. The server administrator may delegate management of specific configuration settings to users without administrative privileges on the server machine.

The file-based configuration for a specific site or application can be copied from one computer to another, for example, when the application moves from development into test and then into production. Due to xcopy deployment of configuration beside code and content, it is significantly easier to deploy applications on IIS 7.0.

Distributed configuration system also enables configuration for a site or application to be shared across a Web server farm, where all servers retrieve configuration settings from a single server. After a Web site is in production, administrators can share configuration information across multiple front-end Web servers, avoiding costly and error-prone replication and manual synchronization issues.

The IIS 7.0 configuration system is fully extensible and allows you to extend the configuration store to include custom configuration. The system is backward compatible with previous versions of IIS at the API level, and with previous versions of the .NET Framework at the XML level.

Note For more information on IIS 7.0 distributed configuration system, refer to Chapter 4, "Understanding the Configuration System."

Administration Tools

IIS 7.0 administration tools have been completely rewritten. They provide different interfaces for reading from and writing to the hierarchy of configuration files on the server, including the applicationHost.config file, the .NET Framework root web.config file, and web.config files for sites, applications, and directories, as well as interfaces for working with run-time information and different providers on the server.

IIS 7.0 provides the following administration tools:

- IIS Manager is a new management console that offers an intuitive, feature-focused, task-oriented graphical user interface (GUI) for managing both IIS 7.0 and ASP.NET. IIS Manager in IIS 7.0 is implemented as a Windows Forms application that replaces the MMC snap-in used in previous versions of IIS.
- A command line tool, Appcmd.exe, replaces IIS 6.0 command line scripts. It provides command line access to configuration files hierarchy and other server settings.
- The Microsoft.Web.Administration interface provides a strongly typed managed API for managed code access to configuration and other server settings.
- A new WMI provider offers scripting access to all IIS and ASP.NET configuration. The legacy IIS 6.0 WMI provider is still available for backward compatibility with existing scripts.

You can also use Windows PowerShell for powerful scripting access to distributed configuration hierarchy.

Note For more information on using PowerShell to manage IIS 7.0, refer to Chapter 7, "Using Command Line Tools."

In addition, the IIS 6.0 MMC snap-in is also provided with Windows Server 2008 to support remote administration and to administer FTP sites.

All new administration tools fully support the new IIS 7.0 distributed configuration, and all of them allow for delegation of access to configuration for individual sites and applications to users without administrative privileges on the server machine.

Note You can install administration tools and Web server components separately.

Figure 1-4 shows the new IIS Manager user interface that has a browser-like feel with an address bar similar to Windows Explorer. The main body of the IIS Manager window is divided into three areas:

- The Connections pane on the left side of the IIS Manager window enables you to connect to servers, sites, and applications. The connections are displayed in a tree.
- A central area referred to as a workspace is located in the middle of IIS Manager window. The workspace has two views: Features View and Content View.
  - Features View enables you to view and configure features for the currently selected configuration path. Each IIS Manager feature typically maps to a configuration section that controls the corresponding Web server feature.

- Content View provides a read-only display of content corresponding to the currently selected configuration path. In Content View, when you select a node in the tree in the Connections pane tree, its content is listed in the workspace.

- An Actions Pane is located on the right side of IIS Manager. Items in the Actions pane are task-based and context-specific.

Figure 1-4 IIS Manager UI.

As with other administration tools, delegated management is one of the most important capabilities of IIS Manager. With this capability, users of hosted services can run IIS Manager on their desktops and connect remotely to manage their sites and applications on the server where they are hosted without having administrative access to the server machine. To identify users, IIS Manager can use Windows credentials and also alternative credentials stores. IIS Manager credentials are particularly useful in scenarios in which you don't want to create Windows accounts for all remote users, or when the credentials are already stored in a non-Windows authentication system and you want to keep them in a single store.

IIS Manager supports remote administration over a firewall-friendly HTTPS connection, allowing for seamless local and remote administration without requiring Distributed Component Object Model (DCOM) or other administrative ports to be opened on the firewall. In IIS 6.0, management console remoting was through the MMC and was always enabled. This is different in IIS 7.0, where remote management through IIS Manager is disabled by default and must be explicitly enabled. For remote administration of IIS 7.0, Web Management Service (WMSvc) must be installed on the server computer, and the remote connections to this service must be enabled. WMSvc is a Windows service that provides the ability to manage IIS 7.0 sites and applications remotely using IIS Manager. IIS Manager remoting architecture is shown in Figure 1-5.

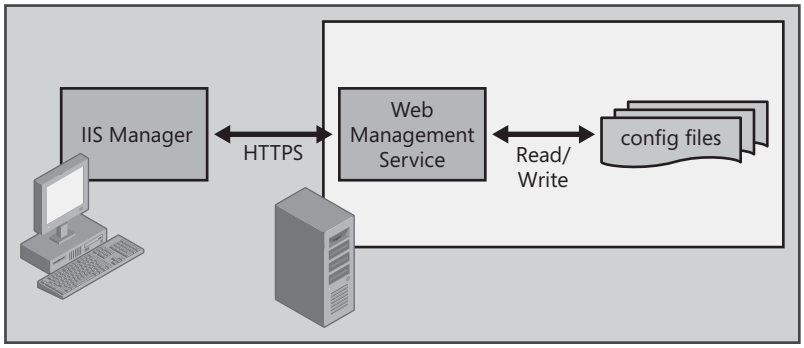


Figure 1-5 IIS Manager remoting.

IIS Manager in IIS 7.0 is customizable and extensible. It has its own configuration file, administration.config, that enables custom functionality to be added to the tool. Any added administration plug-ins are integrated into the tool and appear alongside IIS and ASP.NET features.

Note For more information on IIS Manager, refer to Chapter 6, "Using IIS Manager," and for more information on Appcmd.exe, WMI, and Microsoft.Web Administration API, refer to Chapter 7.

Diagnostics

IIS 7.0 introduces major improvements in diagnostics and troubleshooting of Web sites and applications. It enables you to troubleshoot issues quicker and minimize Web site downtime through powerful new diagnostic capabilities including access to run-time information and automatic tracing of failed requests. The diagnostics and troubleshooting changes in IIS 7.0 enable you to see, in real time, requests that are running on the server and to automatically trap errors with a detailed trace log.

Access to Run-Time Information

IIS 7.0 includes a new Runtime State and Control API (RSCA) that provides real-time state information about application pools, worker processes, sites, application domains, and running requests.

The RSCA is designed to give administrators an in-depth view into the current state of the run-time objects, including current worker processes and their currently executing requests, and also to enable administrators to use the same API to control those objects. RSCA allows administrators to get detailed run-time data that was not previously available.

This information is exposed through a native Component Object Model (COM) API. The API itself is wrapped and exposed through the new IIS 7.0 WMI provider, Microsoft.Web.Administration API, command line management tool Appcmd.exe, and IIS Manager.

For example, using IIS Manager, administrators can get run-time information on what requests are currently executing, how long they have been running, which URLs they are invoking, what client called them, and what their status is.

Failed Request Tracing

IIS 7.0 provides detailed trace events throughout the request and response path, enabling you to trace a request as it makes its way to IIS, through the IIS request processing pipeline, into any existing page-level code, and back out to the response. These detailed trace events enable you to understand not only the request path and any error information that was raised as a result of the request, but also elapsed time and other debugging information to assist in troubleshooting all types of errors and when a system stops responding.

Problems such as poor performance on some requests, authentication-related failures on other requests, or the server 500 error can often be difficult to troubleshoot unless you have captured the trace of the problem when it occurs. That's where failed request tracing can be helpful. It is designed to buffer the trace events for a request and then save them to disk into the trace log if the request fails. To enable the collection of trace events, you can configure IIS 7.0 to automatically capture full trace logs in XML format for any given request based on elapsed time or error response codes.

The diagnostic capabilities in IIS 7.0 are extensible, and new trace events can be inserted into custom modules.

Note For more information on diagnostics and troubleshooting, refer to Chapter 16, "Tracing and Troubleshooting."

Windows Process Activation Service

IIS 7.0 provides a new protocol-independent Windows Process Activation Service (WAS) that is an extended and generalized successor to Windows Activation Service in IIS 6.0. The HTTP process activation model was introduced in IIS 6.0 with application pools. This service has been extended in IIS 7.0 to be available for more than just Web applications. It is capable of receiving requests or messages over any protocol and supports pluggable activation of arbitrary protocol listeners. In addition to being protocol-independent, WAS provides all types of message-activated applications with intelligent resource management, on-demand process activation, health monitoring, and automatic failure detection and recycling. The Windows Communication Foundation (WCF) ships with protocol adapters that can leverage the capabilities of WAS. Using these capabilities can dramatically improve the reliability and resource usage of WCF services.

Note For more information on WAS and non-HTTP support in IIS 7.0, refer to Chapter 2.

Application Compatibility

IIS 7.0 is built to be compatible with previous releases of IIS. Most existing ASP, ASP.NET 1.1, and ASP.NET 2.0 applications are expected to run on IIS 7.0 without code changes, using the compatible ISAPI support.

All existing ISAPI extensions and most ISAPI filters also continue to work. However, ISAPI filters that use READ RAW DATA notification are not supported in IIS 7.0.

For existing Active Directory Service Interfaces (ADSI) and WMI scripts, IIS 7.0 provides feature parity with previous releases, enabling the scripts to use legacy configuration interfaces by using the Metabase Compatibility layer.

Note For more information on application compatibility, see Chapter 11, "Hosting Application Development Frameworks."

Basic Administration Tasks

For a Web server to start serving content, it must have a basic configuration: a site, an application, a virtual directory, and an application pool. IIS 7.0 provides a default configuration that includes the Default Web Site with a root application mapped to a physical directory *%SystemDrive%*\Inetpub\Wwwroot and a default application pool called DefaultAppPool that this application belongs to.

However, you may need to create your own site, add an application to the site, add a virtual directory to the application, create a new application pool, and assign an application to the application pool. The following sections describe how to perform these basic administration tasks by using IIS Manager.

Note For information on how to perform other common administrative tasks, refer to Appendix J, "Common Administrative Tasks Using IIS Manager."

To start IIS Manager, from the Administrative Tools program group, launch Internet Information Services (IIS) Manager.

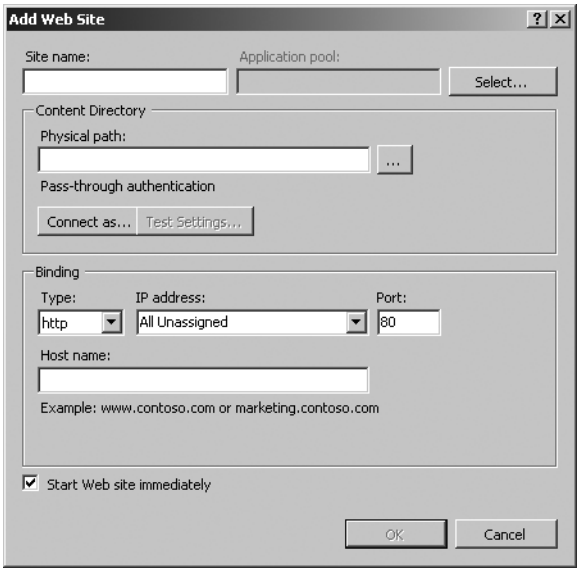
Creating a Web Site

A site is a container for applications and virtual directories. Each site can be accessed through one or more unique bindings. The binding includes the binding protocol and the binding information. The binding protocol defines the protocol over which communication occurs between the IIS 7.0 server and a Web client such as a browser. The binding information defines the information that is used to access the site. For example, the binding protocol of a

Web site can be either HTTP or HTTPS, and the binding information is the combination of IP address, port, and optional host header.

To create a Web site using IIS Manager, perform the following steps:

1. In the Connections pane, expand the server node, right-click the Sites node, and then click Add Web Site. The Add Web Site dialog box appears.



2. In the Site Name box, type a name for your Web site, for example, **www.contoso.com.**
3. If you want to assign a different application pool than the one listed in the Application Pool box, click Select. Then in the Select Application Pool dialog box, choose an application pool from the Application Pool drop-down list and click OK.
4. In the Physical Path box, type the physical path of the Web site's folder or navigate to the folder by using the browse button (...).

   If the physical path that you entered points to a remote share, click Connect As and specify the required credentials. If no credentials are required to access the path, select the Application User (Pass-Thru Authentication) option in the Connect As dialog box.
5. Optional: Click Test Settings to verify the settings you specified.
6. Configure the desired bindings for your new site:
   - If you are using HTTPS for the Web site access, in the Type drop-down list, change the protocol from HTTP to HTTPS.
   - If you have a dedicated static IP address for the site, in the IP Address box, type that IP address. If you don't have a static IP address for the site, leave the default value of All Unassigned.

- If your site will use a different port number than the default port number of 80, in the Port box, type that port number.
- If your site will use a host header, in the Host Name box, type that host header name for your site. For example, type **www.contoso.com.**

7. If you want the Web site to be immediately available, select the Start Web Site Immediately check box.
8. Click OK. The new Web site has been created and appears in the Connections pane.

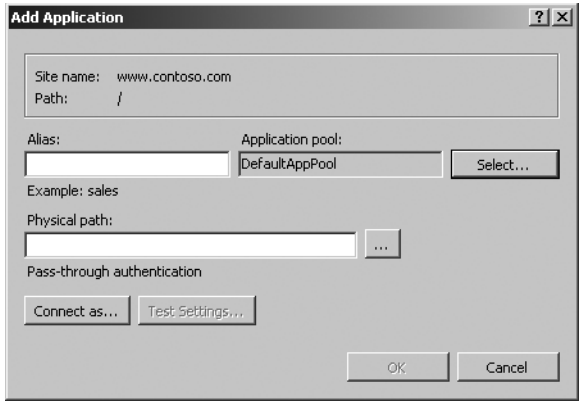
Creating an Application

An application is a group of files that delivers content or provides services over protocols, such as HTTP. When an application is created, the application's path becomes part of the URL.

A site can contain many applications including that site's default application, which is called the root application. In addition to belonging to a site, an application belongs to an application pool, which isolates the application from applications in other application pools on the server.

To create an application using IIS Manager, perform the following steps:

1. In the Connections pane, right-click the site where you want the new application to run. Then select Add Application. The Add Application dialog box appears.



2. In the Alias box, type a value for the application URL, such as **Ads.** This value is used to access the application in a URL.
3. If you want to assign a different application pool than the one listed in the Application Pool box, click Select. Then in the Select Application Pool dialog box, choose an application pool from the Application Pool drop-down list and click OK.
4. In the Physical Path box, type the physical path of the Web site's folder or navigate to the folder by using the browse button (...).

   If the physical path that you entered points to a remote share, click Connect As and specify the required credentials. If no credentials are required to access the path, select the Application User (Pass-Thru Authentication) option in the Connect As dialog box.
5. Optional: Click Test Settings to verify the settings you specified.
6. Click OK. The new application has been created and appears in the Connections pane.

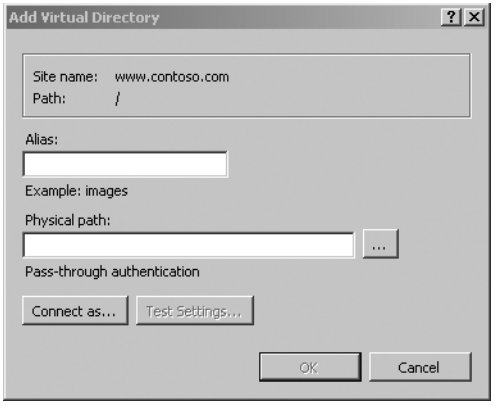
Creating a Virtual Directory

A virtual directory is a directory name (also referred to as path) that is mapped to a physical directory on a local or remote server. That name becomes part of the URL, and a request to this URL from a browser accesses content in the physical directory, such as a Web page or a list of a directory's content.

An application can contain many virtual directories. Each application must have a root virtual directory that maps the application to the physical directory that contains the application's content.

To create a virtual directory using IIS Manager, perform the following steps:

1. In the Connections pane, right-click the site where you want the virtual directory to appear. Then select Add Virtual Directory. The Add Virtual Directory dialog box appears.



2. In the Alias box, type a value for the virtual directory URL, such as **Download.** This value is used to access the application in a URL.
3. In the Physical Path box, type the physical path of the Web site's folder or navigate to the folder by using the browse button (...).

   If the physical path that you entered points to a remote share, click Connect As and specify the required credentials. If no credentials are required to access the path, select the Application User (Pass-Thru Authentication) option in the Connect As dialog box.
4. Optional: Click Test Settings to verify the settings you specified.
5. Click OK. The new virtual directory has been created and appears in the Connections pane.

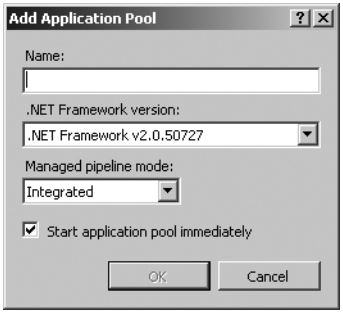
Creating an Application Pool

An application pool is a group of one or more applications that a worker process, or a set of worker processes, serves. Application pools set boundaries for the applications they contain, providing isolation between applications running in different application pools.

In IIS 7.0, ASP.NET requests within application pools can be executed in one of two managed pipeline modes: Integrated or Classic. In Integrated mode, the server uses the unified, or integrated, request processing pipeline to process the request. In Classic mode, the server processes ASP.NET requests using two different IIS and ASP.NET pipelines, in the same way as if the application were running in IIS 6.0.

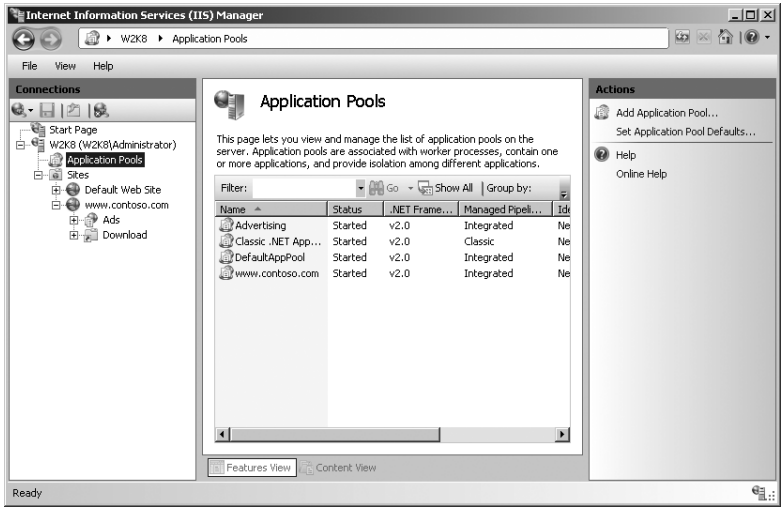
To create an application pool using IIS Manager, perform the following steps:

1. In the Connections pane, expand the server node and right-click the Application Pools node. Select Add Application Pool. The Add Application Pool dialog box appears.



2. In the Name box, type a friendly name for the application pool, for example, **Advertising**.

3. From the .NET Framework Version drop-down list, select the version of the .NET Framework required by your managed applications, modules, and handlers. If the applications that you run in this application pool do not require the .NET Framework, select No Managed Code.
4. From the Managed Pipeline Mode drop-down list, select one of the following options:
   - **Integrated** Select this if you want to use the integrated IIS and ASP.NET request processing pipeline. This is the default mode.
   - **Classic** Select this if you want to use IIS and ASP.NET request-processing modes separately.
5. By default, the Start Application Pool Immediately check box is selected. If you do not want the application pool to start, clear the box.
6. Click OK. The new application pool has been created and appears in the Application Pools list.



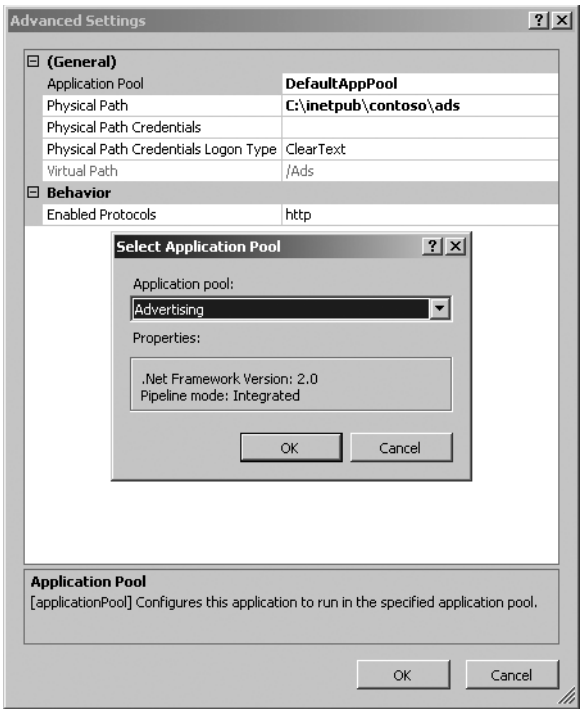
Assigning an Application to an Application Pool

You can assign an application to its own application pool if you want to isolate this application from other applications running on the server. You can assign several applications to the same application pool if all the applications use the same run-time configuration settings, for example, worker process settings or ASP.NET version.

To assign an application to an application pool using IIS Manager, perform the following steps:

1. In the Connections pane, right-click an application you want to assign to a different application pool, select Manage Application, and then click Advanced Settings.

2. On the Advanced Settings page, select Application Pool and then click the browse button. The Select Application Pool dialog box appears.



3. Select the application pool you want the application to run in.
4. Click OK. The application has been assigned to the application pool.

IIS 7.0 Features in Windows Server 2008 and Windows Vista

IIS 7.0 is a part of Windows Server 2008 and Windows Vista. However, the availability of IIS 7.0 features varies between Windows Server 2008 and the editions of Windows Vista.

Windows Server 2008 includes all IIS 7.0 features. IIS 7.0 is available in all editions of Windows Server 2008. There is no difference in functionality among editions. IIS 7.0 is available on 32-bit and 64-bit platforms.

IIS 7.0 is supported in Server Core installations of Windows Server 2008. IIS 7.0 on Server Core provides you with a Web server on top of a minimal footprint server operating system, with a smaller disk space requirement, lower memory utilization, reduced attack surface, and lower servicing needs. IIS 7.0 installation on Windows Server 2008 Server Core is

different from a regular Windows Server 2008 IIS 7.0 installation. On Server Core, there is no Windows shell and .no NET Framework. As a result, IIS Manager is not available, and you cannot run ASP.NET modules, handlers, and applications on Server Core. You can, however, run ASP, PHP, CGI, and other nonmanaged application code on Server Core installations of IIS 7.0.

Note For more information on installing IIS 7.0 on Server Core, refer to Chapter 5, "Installing IIS 7.0."

In Windows Vista editions, IIS 7.0 provides Web developers with a Web platform for building and testing Web applications for IIS 7.0 and also enables process activation and management infrastructure for Microsoft's Windows Communication Foundation (WCF) applications. This infrastructure is provided by Windows Process Activation Service.

The IIS 7.0 features available in a Windows Vista installations depend on the edition of Windows Vista, as follows:

- In Windows Vista Starter and Home editions, IIS 7.0 components only offer supporting infrastructure for WCF but do not provide a Web server that supports static content, Classic ASP, or ASP.NET.
- In Windows Vista Home Premium edition, most of the IIS 7.0 Web Server features required for Web site development are available. However, FTP server, advanced Web authentication and authorization, and remote administration are not available.
- In Windows Vista Business, Enterprise, and Ultimate editions, all of the IIS 7.0 features are available with exception of remote administration.

Table 1-1 lists availability of features in Windows Server 2008 and editions of Windows Vista. Within the table, the features are grouped into categories as follows:

- Common HTTP features
- Application development features
- Health and diagnostics features
- Security features
- Performance features
- Management tools
- Windows Process Activation Service
- File Transfer Protocol (FTP) publishing service features
- Simultaneous connection limits

Within each category, the feature availability is described as follows:

- **Default** The feature is selected by default when you install IIS 7.0. You can decide not to install this feature if you do not need it.
- **Available** The feature is available, but it is not selected by default when you install IIS 7.0. You can install this feature if you need it.
- **Unavailable** The feature is unavailable and cannot be installed when you install IIS 7.0.

Table 1-1 IIS Features in Windows Server 2008 and Windows Vista

Feature Name	Windows Server 2008 Editions	Windows Vista Editions		
		Ultimate, Business, and Enterprise	Home Premium	Home Basic and Starter
Common HTTP Features				
Static Content	Default	Default	Default	Unavailable
Default Document	Default	Default	Default	Unavailable
Directory Browsing	Default	Default	Default	Unavailable
HTTP Errors	Default	Default	Default	Default
HTTP Redirection	Default	Default	Default	Default
Application Development Features				
ASP.NET	Available	Available	Available	Unavailable
.NET Extensibility	Default	Default	Default	Default
ASP	Available	Available	Available	Unavailable
CGI	Available	Available	Available	Unavailable
ISAPI Extensions	Available	Available	Available	Unavailable
ISAPI Filters	Available	Available	Available	Unavailable
Server-Side Includes	Available	Available	Available	Unavailable
Health and Diagnostics Features				
HTTP Logging	Default	Default	Default	Default
Logging Tools	Default	Default	Default	Default
Request Monitor	Default	Default	Default	Default
Tracing	Default	Default	Default	Default
Custom Logging	Available	Available	Available	Unavailable
ODBC Logging	Available	Available	Unavailable	Unavailable
Security Features				
Basic Authentication	Available	Available	Available	Unavailable
Windows Authentication	Available	Available	Unavailable	Unavailable
Digest Authentication	Available	Available	Unavailable	Unavailable
Client Certificate Mapping Authentication	Available	Available	Unavailable	Unavailable

Table 1-1 IIS Features in Windows Server 2008 and Windows Vista

Feature Name	Windows Server 2008 Editions	Windows Vista Editions		
		Ultimate, Business, and Enterprise	Home Premium	Home Basic and Starter
IIS Client Certificate Mapping Authentication	Available	Available	Unavailable	Unavailable
URL Authorization	Available	Available	Available	Available
Request Filtering	Available	Available	Available	Available
IP and Domain Restrictions	Available	Available	Available	Available
Performance Features				
Static Content Compression	Default	Default	Default	Default
Dynamic Content Compression	Available	Available	Available	Available
Management Tools				
IIS Management Console (IIS Manager)	Default	Default	Default	Unavailable
IIS Management Scripts and Tools	Available	Available	Available	Available
Management Service	Available	Available	Available	Unavailable
IIS 6.0 Management Compatibility	Available	Available	Available	Available
IIS Metabase Compatibility	Available	Available	Available	Available
IIS 6 WMI Compatibility	Available	Available	Available	Unavailable
IIS 6 Scripting Tools	Available	Available	Available	Unavailable
IIS 6 Management Console	Available	Available	Available	Unavailable
Windows Process Activation Service Features				
Process Model	Default	Default	Default	Default
.NET Environment	Available	Available	Available	Available
Configuration APIs	Available	Available	Available	Available
File Transfer Protocol (FTP) Publishing Service Features				
FTP Server	Available	Available	Unavailable	Unavailable
FTP Management Console	Available	Available	Unavailable	Unavailable
Simultaneous Connection Limits				
Simultaneous Connection Limits	Unlimited	10	3	3

Summary

IIS 7.0 has been completely redesigned and re-engineered from the ground up. IIS 7.0 offers major advantages over previous versions of IIS and makes developing, deploying, and configuring and managing Web applications and infrastructure easier and more efficient than ever before.

IIS 7.0 delivers many new powerful features and functionality based on the following key enhancements:

- **Modularity** IIS 7.0 architecture is fully componentized. It enables administrators to customize which features are installed and running on the Web server. With more than 40 feature modules that can be independently installed, administrators can reduce the potential attack surface and lower the footprint requirements of the server.
- **Extensibility** The core Web server features of IIS 7.0 have been built using a new set of comprehensive public APIs that developers can use to extend, replace, or add functionality to a Web server. These APIs are available as native Win32 APIs as well as managed .NET Framework APIs. Developers can also extend IIS configuration and build IIS Manager extensions that plug in seamlessly to the management console.
- **Unified distributed configuration system** IIS 7.0 provides a unified distributed file-based configuration system for storing all IIS and ASP.NET settings in a single clear-text XML format in a configuration files hierarchy where configuration files are stored together with Web site and application content. This configuration system enables xcopy deployment of configuration alongside application code and content, and it also provides an easy way to share a configuration across a Web farm.
- **New administration tools** IIS 7.0 offers a set of administration tools that simplify managing Web infrastructure and allow administrators to delegate administrative control for sites and applications to developers and content owners. IIS 7.0 includes a new GUI management console, IIS Manager; a new command line utility, Appcmd.exe; a new WMI provider for automating administration tasks; and a new managed API. All of these tools provide unified support for managing IIS and ASP.NET settings together. Administrators and developers can also use Windows PowerShell for scripting access to configuration information for the entire Web platform.
- **Integrated diagnostics** IIS 7.0 enables administrators and developers to minimize downtime by using new diagnostics and troubleshooting capabilities. IIS 7.0 exposes run-time diagnostic information including currently executing requests. IIS 7.0 can also be configured to automatically log detailed trace events for failed requests for errant Web sites and applications.

Additional Resources

These resources contain additional information and tools related to this chapter:

- For more information on IIS 7.0 request processing, refer to Chapter 2, "Understanding IIS 7.0 Architecture."
- For more information about modularity, refer to Chapter 3, "Understanding the Modular Foundation."
- For more information on IIS 7.0 extensibility, refer to Chapter 12, "Managing Web Server Modules," and Chapter 13, "Managing Configuration and User Interface Extensions."
- For more information about the unified distributed configuration system, refer to Chapter 4, "Understanding the Configuration System."
- For more information about administration tools, refer to Chapter 6, "Using IIS Manager," and Chapter 7, "Using Command Line Tools."
- For more information about the troubleshooting capabilities of IIS 7.0 and how to use them, refer to Chapter 16, "Tracing and Troubleshooting," and Chapter 17, "Performance and Tuning."

Chapter 4

Understanding the Configuration System

In this chapter:

On the Disc Browse the CD for additional tools and resources.

Many of the new features and capabilities of Internet Information Services (IIS) 7.0 can be attributed to its entirely new configuration system. The metabase of old has been transformed into a .NET configuration–inspired system that is much easier on many levels to support. The new design provides the basis for delegated configuration, centralized configuration, ASP.NET integration, xcopy deployment of configuration, and many other benefits.

In many cases, the IIS 7.0 configuration system will "just work," and you won't need to know what's going on behind the scenes. However, when you add flexibility to a system, you often introduce complexity, which is the case with the IIS 7.0 configuration system. This chapter details the configuration's operation so that you'll have a thorough understanding of what's going on.

As shown in Figure 4-1, the configuration of IIS 7.0 as a whole is composed of several systems that work both together and independently. For administrators with an understanding of the .NET configuration files and how they work, IIS 7.0 configuration is a quick study. If your only exposure to IIS configuration has been using a tool such as Metabase Explorer, then there's a bigger—but worthwhile—learning curve.

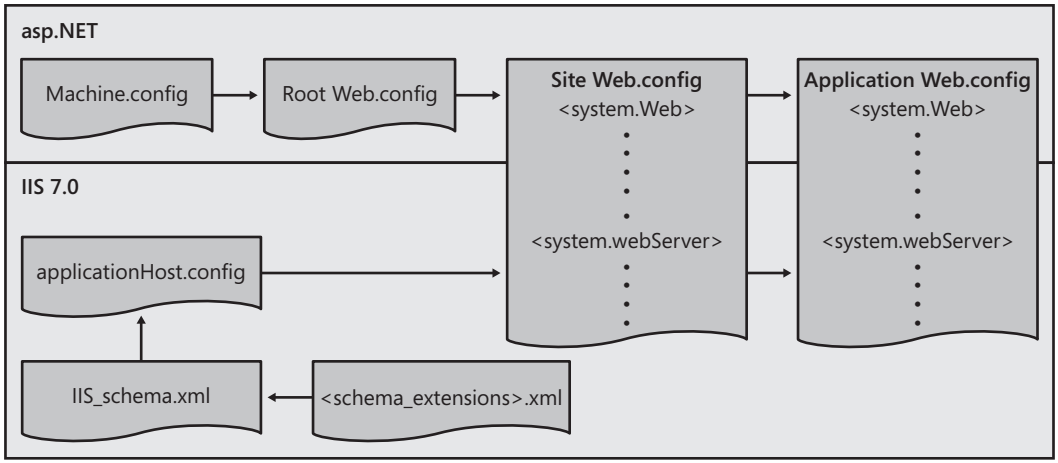


Figure 4-1 The IIS 7.0 configuration system.

Overview of the Configuration System

The IIS 7.0 configuration system is in many ways a complete departure from the metabase, the configuration model that previous IIS versions use. The new architecture reflects requirements that the IIS 7.0 configuration system be more manageable and flexible in supporting key deployment scenarios.

The IIS 7.0 configuration system is based on a hierarchy of XML configuration files, which contain structured XML data that describes the configuration information for IIS and its features. This hierarchy includes the .NET Framework configuration files, machine.config and root web.config; the main IIS configuration file called applicationHost.config; and distributed web.config configuration files located inside the Web site directory structure. One key benefit of this hierarchy is the ability to unify the location of IIS and ASP.NET configuration information. The other is the ability to include IIS configuration together with the Web site's content, which makes the Web site portable and alleviates the need to have administrative privileges to deploy the Web site.

The configuration files in the hierarchy contain configuration sections, which are structured XML elements that describe configuration settings for specific IIS features. Unlike the property/value model used by the metabase, the structured XML nature of the IIS 7.0 configuration sections helps the configuration become cleaner and easier to understand. This makes configuration self-explanatory, and you can easily edit it by hand. For example, the application developer can place the following configuration in a web.config file located in the root of the Web site to enable the IIS default document feature and configure a specific default document to be used.

```
<system.webServer>
  <defaultDocument enabled="true">
    <files>
      <add value="home.aspx" />
```

```
    </files>
  </defaultDocument>
</system.webServer>
```

Because the IIS 7.0 configuration system uses the same web.config files as the ASP.NET configuration system, your application can provide both ASP.NET and IIS configuration settings side by side in the same file. Because this file travels with your application content, it enables the application to be deployed to an IIS server simply by copying its contents, without having to modify any central configuration.

At the same time, the server administrator can place server-level IIS configuration, such as the Web site and Application pool definitions, in the server-level applicationHost.config file. This file can also contain the default configuration for other IIS sections, which are by default inherited by all Web sites on the server. Unlike the Web site's web.config files, which may be accessible to the Web site or application administrator, applicationHost.config is accessible only to the server administrator. Using the configuration-locking mechanisms that the configuration system provides, the administrator can specify which configuration can be modified by applications through the use of distributed web.config files.

All in all, the new configuration file hierarchy offers a lot more flexibility than the IIS 6.0 metabase and enables key deployment and management scenarios. Next, we will look at how the configuration file hierarchy works and the syntax of configuration sections.

Configuration File Hierarchy

The metabase in previous versions of IIS was comprised of a single configuration file, Metabase.xml, that contained a URL-centric configuration tree. Nodes of that tree corresponded to URLs on the server, and each node contained a set of properties that specified the configuration for that URL along with properties inherited from parent nodes. If you are familiar with the IIS 6.0 metabase, you may remember that these nodes are addressed via paths that look something like "LM\W3SVC\1\ROOT", which translates to "the root of the Web site with ID of 1."

In IIS 7.0, configuration file hierarchy includes multiple configuration files. Instead of encoding the entire URL hierarchy in a single file, the configuration file hierarchy maps to the URL hierarchy. Each file defines configuration that is implicitly associated with a specific URL level based on the position of this file in the configuration hierarchy. For example, applicationHost.config contains global settings applying to all sites on the server, and web.config, contained in the Web site root, is site-specific—and when contained in an application directory, it is directory-specific. Web.config typically maps to a URL such as *http://www.contoso.com/appfolder*. Note that the use of web.config to contain distributed configuration information is optional (but enabled by default for certain settings). ApplicationHost.config can and often does contain site- and application-specific settings. There are other configuration files involved with IIS 7.0 that we will discuss later in the chapter, but for the sake of simplicity, we'll focus on the files used to configure sites and applications, as listed in Table 4-1.

Table 4-1 IIS 7.0 Configuration Files

File	Location	Configuration Path
machine.config	%windir%\Microsoft .NET\Framework \ <version>\config	MACHINE
root web.config	%windir%\Microsoft .NET\Framework \ <version>\config	MACHINE/WEBROOT
applicationHost.config	%windir%\system32\ inetsrv\config	MACHINE/WEBROOT/APPHOST
distributed web.config files	Web site directory structure	MACHINE/WEBROOT/APPHOST /<SiteName>/<VirtualPath>

Just like the metabase, the IIS 7.0 configuration system uses a *configuration path* to describe the level in the configuration hierarchy where a particular configuration setting is set. This level corresponds both to the URL namespace at which the configuration is effective and a configuration path used in commands (such as when using Appcmd) to reference the correct configuration store. In this way, the IIS 7.0 configuration file hierarchy maps to the URL namespace and can correspond to an actual configuration file where this configuration is set.

When the configuration system retrieves configuration for a specific configuration path, it merges the contents of each configuration file corresponding to each segment of the path, building an effective configuration set for that path. This works well with the ability to specify distributed web.config files inside the Web site's directory structure, which may enable any part of the Web site to set specific configuration for its URL namespace simply by including it in a web.config file in the corresponding directory.

In this system, the configuration path for a particular URL becomes MACHINE/WEBROOT/APPHOST/<SiteName>/<VirtualPath>, where the <SiteName> is the name of the site and the <VirtualPath> is the URL's virtual path. When reading configuration for this path, the server will merge the configuration in machine.config, root web.config, applicationHost.config, and all distributed web.config files that exist in the physical directories corresponding to each segment of the virtual path, starting with the site's root.

Important The root web.config corresponding to WEBROOT in the configuration system is the one located in %windir%\Microsoft .NET\Framework \<version>\config. This is not the same as a web.config file that can placed in a Web site's home directory, which is often referred to as the *web root*. In the first case, we are talking about web.config used by .NET that is the parent, or root of all Web site web.config files. In the latter case, we're talking about the web.config found in a Web site's home folder. The web.config in the Web site's home folder will inherit configuration settings found in the .NET root web.config.

Server-level configuration for IIS features is stored in the applicationHost.config file. This file stores configuration for sections that only make sense globally on the server, as well as

configuration defaults for other sections that are inherited by all URLs on the server unless another file lower in the configuration hierarchy overrides them.

For example, if you wanted to configure the server to disable directory browsing by default, you would put that configuration in the applicationHost.config file. Then, if you wanted to allow directory browsing for the /App1 application in the default Web site, you would place a web.config file containing configuration that enables directory browsing in the physical directory root of the /App1 application. When a request is made to the root of the default Web site, the server will read configuration for the "MACHINE/WEBROOT/APPHOST/Default Web Site/" path and apply the inherited configuration from applicationHost.config that disables the directory browsing. However, when an HTTP request is made to the /App1 application, the server will read configuration for "MACHINE/WEBROOT/APPHOST/Default Web Site/App1/", which merges the configuration set by the application's web.config and enables directory browsing for that URL.

machine.config and root web.config

Even though machine.config and the root.web.config are .NET Framework configuration files, they are read and mapped in by the IIS configuration system. This allows IIS 7.0 to share its configuration with ASP.NET in site and application web.config files, consume .NET modules in the managed pipeline, and integrate .NET configuration that is enabled in the IIS Manager. As previously mentioned, machine.config contains machine-wide .NET Framework configuration settings loaded by all .NET applications on the machine, and root web.config contains ASP.NET-specific configuration settings loaded by all ASP.NET applications. These files are modifiable only by machine administrators.

These files are located in the %windir%\Microsoft .NET\.NET Framework \<version>\config, where the <version> is determined by the *managedRuntimeVersion* setting for the application pool within which the configuration is being read. This way, IIS application pools that are set to use different versions of the .NET Framework automatically include the configuration files for the right .NET Framework version. Note that as in IIS 6.0, an application pool cannot host more than one version of the .NET Framework.

applicationHost.config

The main IIS configuration file is applicationHost.config, which is located in the %windir%\system32\ inetsrv\config directory. It is modifiable only by machine administrators.

ApplicationHost.config contains configuration sections and settings that only make sense globally on the server. For example, it contains site, application, and virtual directory definitions in the <sites> section and the application pool definitions for the <applicationPools> section. Other global sections include the <globalModules> configuration section, which contains a list of native modules that are loaded by all IIS worker processes, and the <httpCompression> section that lists enabled compression schemes and content types that can be compressed.

These sections cannot be overridden at lower levels, and the server only reads them at the MACHINE/WEBROOT/APPHOST level.

ApplicationHost.config also stores all of the default settings for IIS configuration sections, which are inherited by all other URLs unless another configuration file lower in the configuration hierarchy overrides them. In fact, if you examine the contents of applicationHost.config, you will see that it declares all IIS configuration sections.

```
    <configSections>
        <sectionGroup name="system.applicationHost">
            <section name="applicationPools" allowDefinition="AppHostOnly"
overrideModeDefault="Deny" />
            <section name="sites" allowDefinition="AppHostOnly"
overrideModeDefault="Deny" />
            <section name="webLimits" allowDefinition="AppHostOnly"
overrideModeDefault="Deny" />
            …
        </sectionGroup>

        <sectionGroup name="system.webServer">
            <section name="asp" overrideModeDefault="Deny" />
            <section name="caching" overrideModeDefault="Allow" />
            <section name="cgi" overrideModeDefault="Deny" />
            <section name="defaultDocument" overrideModeDefault="Allow" />
            <section name="directoryBrowse" overrideModeDefault="Allow" />
            …
        </sectionGroup>
    </configSections>
```

You may notice that these section definitions include an element named *allowDefinition* that is set in our example to "AppHostOnly". The *allowDefinition* settings assign a scope to the section that limits where the section can be used. In this case, the Sites section can only be used in applicationHost.config and is not legal in any other location. It is strongly recommended that you do not edit the *allowDefinition* settings from the defaults.

Finally, this file also contains information about which configuration sections are allowed to be overridden by lower configuration levels, and which are not. Child override is controlled by the *overrideModeDefault* attribute in the example just provided of the configuration sections declarations. The server administrator can use this attribute to control the delegation of IIS features to the site administrators. We will review controlling section delegation in the Delegating Configuration section of this chapter.

Distributed web.config Files

The IIS 7.0 configuration hierarchy enables the site directory structure to contain web.config configuration files. These files can specify new configuration settings or override configuration settings set at the server level for the URL namespace corresponding to the directory where they are located (assuming the configuration sections used are unlocked by the administrator).

This is the foundation for the delegated configuration scenario, which enables applications to specify required IIS settings together with their content, and which makes simple xcopy deployment possible.

Finally, because the ASP.NET configuration system also reads these files, they can contain both IIS and ASP.NET configuration settings.

redirection.config

You will also find redirection.config located in the %windir%\system32\ inetsrv\config directory, and it is used to store configuration settings for Shared Configuration. It is not part of the IIS 7.0 configuration hierarchy, but the configuration system uses it to set up redirection for the applicationHost.config file.

When in use, it specifies the location and access details required for IIS 7.0 to load applicationHost.config from a remote network location, instead of the local inetsrv\config directory. This enables multiple IIS 7.0 servers to share a central configuration file for ease of management. You can learn more about shared configuration in the "Sharing Configuration Between Servers" section of this chapter.

administration.config

The IIS Manager tool uses administration.config (also not part of the IIS 7.0 configuration hierarchy) exclusively to specify its own configuration. It is also located in the %windir%\system32\ inetsrv\config directory.

Among other things, administration.config contains the list of IIS Manager extensions that the tool loads. These extensions provide the features you see in the IIS Manager. Like IIS, the IIS Manager is fully extensible. You can learn more about the extensibility model provided by IIS Manager and how its extensions are configured in Chapter 12, "Managing Web Server Modules."

Temporary Application Pool .config Files

One of the new IIS 7.0 features is enhanced Application Pool Isolation. At run time, IIS 7.0 reads applicationHost.config configuration and generates filtered copies of it for each application pool, writing them to:

```
%systemdrive%\inetpub\temp\appPools\<ApplicationPoolName>.config
```

The filtered configuration files contain only the application pool definitions for the current application pool (other application pool definitions that may contain custom application pool identities are filtered out). Also removed are all site definitions and site-specific configuration specified in location tags for sites that do not have applications in the current application pool.

The temporary configuration file created for each application pool is protected in such a way that only the application pool for which it is created can read the file. This ensures that no worker process (application pool) can read the configuration settings for any other worker process.

The application pool configuration files are not intended to be used for updates, and neither administrators nor developers should edit them directly or indirectly. Their use is completely transparent, but it is part of the configuration system, so we thought it should be called out here. For more details, see Chapter 14, "Implementing Security Strategies."

Configuration File Syntax

Each configuration file uses special XML elements called *configuration sections* to specify configuration information. A configuration section is the basic unit of configuration, typically defining the behavior of a specific part or feature in the Web server.

Here is an example of a configuration file that specifies multiple configuration sections:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>

    <system.webServer>

        <asp>
            <cache diskTemplateCacheDirectory="%SystemDrive%\inetpub\temp\
ASP Compiled Templates" />
        </asp>
        <defaultDocument enabled="true">
            <files>
                <add value="index.html" />
                <add value="default.aspx" />
            </files>
        </defaultDocument>

        <directoryBrowse enabled="false" />

    </system.webServer>
<configuration>
```

As you can see, this is a well-formed XML file, with a mandatory root <configuration> element that contains multiple subelements. These subelements are either configuration section elements directly, or *section group* elements such as <system.webServer>. Section groups do not define any settings, they simply group related section elements together. For example, all of the IIS Web server features are under the <system.webServer> section group. Sections are the elements, shown in bold, that contain specific configuration settings.

The configuration section elements each follow a specific structure defined by their schema, which controls what attributes and child elements are allowed inside the section, the type of data they can contain, and various other configuration syntax restrictions. The schema

information is provided inside configuration schema files registered with the IIS 7.0 configuration system. Unlike the ASP.NET configuration system, which uses code to define the structure of its configuration, the IIS 7.0 configuration system is based entirely on declarative schema information. We will examine this schema mechanism a little later in the chapter.

In addition to section groups and configuration sections themselves, configuration files can also contain *section declarations* and *location tags*. Section declarations are necessary to declare a particular section before it can be used, and they also indicate what section group the section belongs to. Location tags enable configuration to be scoped to a specific configuration path, rather than to the entire namespace to which the current configuration file corresponds.

Direct from the Source: Working Around Limits on web.config File Size

By default, the IIS 7.0 configuration system enforces a limit of 100 KB on the file size of web.config files. This is for security purposes, to avoid possible denial-of-service attacks on the server by providing very large configuration files.

In most cases, this size should be sufficient for most situations, but what if your configuration file is bigger than 100 KB? This can happen for applications that use web.config files extensively to store custom configuration. To allow these larger files, you can override the maximum limit by adding a registry key. Create the following key.

```
HKLM\Software\Microsoft\InetStp\Configuration
```

Then create a DWORD value.

```
MaxWebConfigFileSizeInKB
```

Set this value to the file size in kilobytes (make sure you select Decimal when entering the value) to set this as a new machine-wide limit on web.config file size.

Section Declarations

Each section that is used in a configuration file contains a *section declaration* in applicationHost.config. Section declarations are generally created during the installation of the feature and do not typically need to be added manually. For example, following is an excerpt from the applicationHost.config configuration file that declares all IIS configuration sections.

```
    <configSections>
        <sectionGroup name="system.applicationHost">
            <section name="applicationPools" allowDefinition="AppHostOnly"
overrideModeDefault="Deny" />
            <section name="sites" allowDefinition="AppHostOnly"
overrideModeDefault="Deny" />
        </sectionGroup>
        <sectionGroup name="system.webServer">
```

```
            <section name="asp" overrideModeDefault="Deny" />
            <section name="defaultDocument" overrideModeDefault="Allow" />
            <section name="directoryBrowse" overrideModeDefault="Allow" />
            <sectionGroup name="security">
                <sectionGroup name="authentication">
                    <section name="anonymousAuthentication" overrideModeDefault="Deny" />
                    <section name="basicAuthentication" overrideModeDefault="Deny" />
                </sectionGroup>
                <section name="authorization" overrideModeDefault="Allow" />
            </sectionGroup>
        </sectionGroup>
    </configSections>
```

This fragment defines a number of IIS configuration sections, including the global <sites> and <applicationPools> sections read by WAS, and various sections for Web server features, including <asp> and <anonymousAuthentication>. You'll also notice that these sections are nested within the appropriate section groups. Section declarations can specify a number of properties that control where the section is available, including *allowDefinition,* which determines at which level in the configuration hierarchy the section can be used, and *overrideModeDefault,* which determines if lower configuration levels can use the section by default. After the section is declared, it can be used in the current configuration file or anywhere lower in the configuration file hierarchy, meaning it does not need to be re-declared in configuration files below (re-declaring this section will actually result in a configuration error). In fact, all IIS configuration sections are declared in applicationHost.config and therefore are available in any Web site web.config configuration file. The *allowDefinition* and *overrideModeDefault* attributes control the actual ability to use this configuration section in the lower levels.

Section Groups

You use *section group* elements to group related configuration sections together. When you declare each section, it specifies which section group it belongs to by placing its <section> element within the corresponding <sectionGroup> element. This implicitly declares the section group itself. Section groups cannot define any attributes and therefore do not carry any configuration information of their own. Section groups can be nested within one another, but sections cannot. Think of section groups as a namespace qualification for sections.

When specifying the configuration section, you must place it inside the section group element according to the declaration. For example, when providing configuration for the <authorization> section, which is declared in the <system.webServer>/<security> section group, the configuration section must be nested in the corresponding section group elements as follows.

```
<configuration>
    <system.webServer>
        <security>
            <authorization bypassLoginPages="true" />
        </security>
    </system.webServer>

</configuration>
```

Table 4-2 lists most of the section groups you will find in the IIS 7.0 configuration system by default, what configuration they contain, and where they are declared.

Table 4-2 Section Groups

Section Group	Description	Declared In
system.applicationHost	Contains global protocol-neutral IIS configuration used by the Windows Process Activation Service, including <sites>, <applicationPools>, <listenerAdapters>, and more	applicationHost.config
system.webServer	Contains all configuration for the IIS Web server engine and features, including <modules>, <handlers>, <serverRuntime>, <asp>, <defaultDocument>, and dozens more; also contains several child section groups	applicationHost.config
system.webServer /security	Contains security-related Web server configuration, including <authorization>, <isapiCgiRestriction>, <requestFiltering> and more	applicationHost.config
system.webServer /security /authentication	Contains configuration for all authentication Web server features, including <anonymousAuthentication>, <windowsAuthentication>, and more	applicationHost.config
system.webServer /tracing	Contains configuration for tracing Web server features, including <traceFailedRequests> and <traceProviderDefinitions>	applicationHost.config
system.web	Contains all ASP.NET configuration	Framework machine.config

Not listed in Table 4-2, for the sake of brevity, are section groups declared in .NET's machine.config. These sections control various aspects of the .NET Framework behavior, including system.net, system.xml.serialization, and others.

Sections

The *configuration section* is the focus of the IIS 7.0 configuration system, because it is the basic unit of configuration. Each configuration section has a specific structure defined by its schema, containing specific attributes, elements, and collections of elements necessary to express the required configuration for the corresponding IIS feature.

A configuration section may contain 0 or more of the elements (depending on the schema) shown in Table 4-3.

Table 4-3 Configuration Section Elements

Element	Description
Attributes	A named XML attribute, using a type specified in the schema. Supported types include int, string, timespan, enumerations, and others. Attributes may have associated validation rules, which restrict the allowed values. They may also have additional metadata such as default values, or they may specify whether or not the attribute must be specified when the section is used.
Child elements	Child XML elements, which in turn can contain attributes and other child elements.
Collections	A collection is a child element that can contain a list of other child elements (typically <add>, <remove>, and <clear>) that can be used to create lists of configuration items. Collection elements have metadata associated with them that define their behavior, including what attributes serve as collection item keys, the order in which collection items are added when collections are merged between configuration files, and more.

Most configuration sections specify default values for all of the attributes in their schema. This becomes the default configuration for that section if it's not defined in any configuration file (by default, collections are always empty). Each configuration file can specify the section element to explicitly set the value of one or more attributes, or modify the collections in the section. The section can be specified at multiple configuration files, in which case when the configuration system retrieves the contents of this section for a particular configuration path, it merges the contents of all instances of this section. Merging attributes overrides the values specified in the configuration levels above, and merging collections adds/removes/clears items in collections based on the usage of collection elements.

For example, here are the contents of a web.config file that you could place in the root of a PHP application. The contents contain the configuration for the <defaultDocument> section and enable the index.php page to serve as a default document.

```
<configuration>
  <system.webServer>
    <defaultDocument enabled="true">
      <files>
        <add value="index.php" />
      </files>
    </defaultDocument>
  </system.webServer>
</configuration>
```

This configuration overrides the global *enabled* attribute set in applicationHost.config or a higher order web.config, setting its value to "true". It also adds a new item to the <*files*> collection to enable "index.php" to serve as a default document. If configuration files earlier in the hierarchy defined other default document types in the <*files*> collection, then the effective collection for your application would contain those items plus the item we just added at our scope. Likewise, if the parent configuration files disabled the default document feature by setting its *enabled* attribute to "false", our configuration will override that value for the application.

The section titled "Editing Configuration" later in this chapter discusses setting configuration by specifying configuration sections.

Configuration Section Schema

All IIS configuration sections are defined in the IIS_Schema.xml file located in a schema file in the %windir%\system32\inetsrv\config\schema directory. To learn more about the syntax of each configuration section, you can review its schema. For example, here is an excerpt from the schema definition for the <defaultDocument> configuration section.

```
  <sectionSchema name="system.webServer/defaultDocument">
    <attribute name="enabled" type="bool" defaultValue="true" />
    <element name="files">
      <collection addElement="add" clearElement="clear" removeElement="remove"
mergeAppend="false">
        <attribute name="value" type="string" isUniqueKey="true"/>
      </collection>
    </element>
  </sectionSchema>
```

The schema contains the definitions for the "enabled" attribute and the <files> collection that we used earlier to set default document configuration. As you can see, the schema contains more information than just the structure of the configuration section—it also contains various metadata about the format and behavior of attributes and collections, including the types for attributes and which attributes serve as unique keys for collections. The <defaultDocument> section is a fairly simple section, so it doesn't fully illustrate the flexibility of section schema information, but it is a good example of how you can use the schema information to define configuration sections and control their behavior.

Note When working with IIS configuration, you will likely never have to work with section schema. However, it is useful to know where the schema information is located if you need a reference for the structure and semantics of IIS configuration sections. You should *never* attempt to modify the IIS schema files. However, if you are developing new IIS features, you can publish custom configuration schema files into the inetsrv\config\schema directory in order to use new configuration sections with the IIS configuration system.

In the schema directory, you will also find the FX_schema.xml and ASPNET_schema.xml files, which contain the schema definitions for .NET Framework and ASP.NET configuration sections respectively.

The IIS 7.0 configuration system is fully extensible. Custom configuration sections registered with the IIS 7.0 configuration schema will have their own schema files published in the schema directory.

Location Tags

By default, configuration specified in a particular configuration file applies to the entire URL namespace corresponding to that file. For example, configuration set in applicationHost.config applies to the entire server, and configuration set in the site's root web.config file applies to the entire site (unless overridden by more specific web.config files). This works most of the time. However, in some cases it is necessary to apply configuration to a specific subset of the URL namespace, or to a specific URL. *Location tags* are the mechanism that enables this by specifying a configuration path for which all configuration specified within a location tag applies.

Here is an example of using a location tag to scope configuration to a specific Web site.

```
<location path="Default Web Site">
    <system.webServer>
        <directoryBrowse enabled="true" />
    </system.webServer>
</location>
```

This location tag, when specified in applicationHost.config, applies the <directoryBrowse> configuration section to the "MACHINE/WEBHOST/APPHOST/Default Web Site/" configuration path.

You can find Location tags in use with three common scenarios in IIS 7.0:

1. Defining site-specific directory or file configuration in applicationHost.config. This is necessary to apply specific configuration for a content in a Web site without defining it in the site's web.config. For example, this is the technique commonly used by shared hosting servers to set site-specific configuration without giving the site administrators control over that configuration. When making changes to configuration in the IIS Manager or one of the programmatic interfaces, if a setting is not delegated, it is written to applicationHost.config by using location tags.
2. Locking or unlocking a specific configuration section for a particular configuration path. By placing a configuration section inside the location tag for a particular path, you can use the *overrideMode* attribute on the location tag to lock or unlock this configuration section for that path. For example, this is necessary for configuration sections declared with overrideModeDefault = Deny so that you can allow delegated configuration in web.config files.
3. Specifying configuration for a specific nonphysical URL. If you need to apply specific configuration to a URL that does not correspond to a physical directory (a file or a virtual URL), it's necessary to define it using a location tag inside a physical parent directory.

   You can use a location tag to keep all of the configuration for a site or application in a single web.config file, instead of placing pieces of it in many different web.config files in various subdirectories.

We will discuss using location tags in more detail later in this chapter.

The IIS 7.0 Configuration System and the IIS 6.0 Metabase

So far, we've been discussing in some detail the contents and mechanics of the configuration system, but we should back up a bit and discuss applicationHost.config itself rather than its contents.

Differences Between the IIS 7.0 Configuration System and the IIS 6.0 Metabase

The IIS 6.0 configuration store is Metabase.xml and is stored in %windir%\system32\inetsrv. For IIS 7.0, Metabase.xml is transformed into applicationHost.config located in %windir%\system32\inetsrv\config.

Why did the IIS team invest such time and effort in a wholesale change to the structure and mechanics of the configuration system? Primarily to make a quantum leap in performance, scale, and manageability. The IIS 6.0 configuration system is based on a system conceived and implemented with IIS 4.0 that was part of Windows NT. It was time to rebuild with a new set of design criteria.

The resulting system is quite a bit more complex because it is very ambitious. Yet at the same time, it is more manageable, scalable, and flexible. Table 4-4 compares some of the key differences between the IIS 6.0 metabase and the IIS 7.0 configuration files.

Table 4-4 Metabase.xml Comparison to IIS 7.0 Configuration System

Feature	IIS 6.0 Metabase.xml	IIS 7.0 Configuration System	Why This Matters
Delegated configuration	Not possible—all configuration is centrally stored and requires Administrative privileges to change	Enables both administrator-controlled configuration in applicationHost.config and delegated configuration in web.config files	Administrators can delegate configuration tasks to application owners; applications can be xcopy-deployed with all of their configuration
Structural organization	Properties are not grouped	Provides a hierarchy of section groups, sections, elements, and subele-ments	Easy to read, search, and manage; enables use of shorter element name because each item is logically grouped in a section rather than in a flat listing
Simplified description of properties with multiple values	Uses multi-sz key types and bit masks to handle multiple element values such as NT Authentication-Providers	Uses collections with simple add/remove/clear syntax based on .NET Framework configuration syntax and usage	Easier to read, edit, and query settings that can have multiple values

Table 4-4 Metabase.xml Comparison to IIS 7.0 Configuration System

Feature	IIS 6.0 Metabase.xml	IIS 7.0 Configuration System	Why This Matters
Memory vs. file-based configuration	Metabase is a memory construct that is written to Metabase.xml; synchronization issues can occur	Configuration is file-based; configuration writes are persisted directly to the configuration files	IIS configuration is always fully represented in .config files
Schema extensibilty	Difficult to extend for use with custom apps; inhibits innovation from the community	Based on IIS_Schema.xml; schema easily extended with XML snippets	Enables application developers to easily integrate application settings into IIS 7.0

IIS 6.0 Metabase Compatibility

Despite the complete overhaul of the configuration system, IIS 7.0 continues to maintain backward compatibility with existing configuration scripts and tools that target the metabase for configuring the server.

This is accomplished by providing a metabase emulation layer that enables the metabase APIs, exposed through the Active Base Objects (ABO) interfaces on which all other metabase tools and scripting APIs are based. The metabase emulation layer, called the ABO Mapper, provides immediate translation of the metabase configuration structure and actions triggered by callers to the new configuration system. This maps all writes and reads to the metabase to the corresponding IIS 7.0 configuration.

This service performance is transparent to the caller so that the existing installers, configuration scripts, and tools continue to work as if they were working on IIS 6.0. The ABO Mapper makes a best-effort attempt to map all IIS 6.0 metabase properties to the corresponding IIS 7.0 configuration properties that have a known mapping. In the end, virtually all metabase properties can be successfully mapped to the IIS 7.0 configuration, with rare exceptions.

Note You can find documentation that describes how IIS 6.0 metabase properties map to the new IIS 7.0 configuration schema at *http://msdn2.microsoft.com/en-us/library/aa347565.aspx*.

Metabase compatibility is not enabled by default, and you don't need it if you are not running any legacy IIS 6.0 configuration scripts or using third-party installers that require ABO. If you are, though, you will need to install the IIS 6.0 Metabase Compatibility component from the IIS/Metabase Compatibility category in the Turn Windows Features On And Off page of Control Panel\Programs And Features on Windows Vista, or the IIS role in the Server Manager tool on Windows Server 2008, as shown in Figure 4-2.

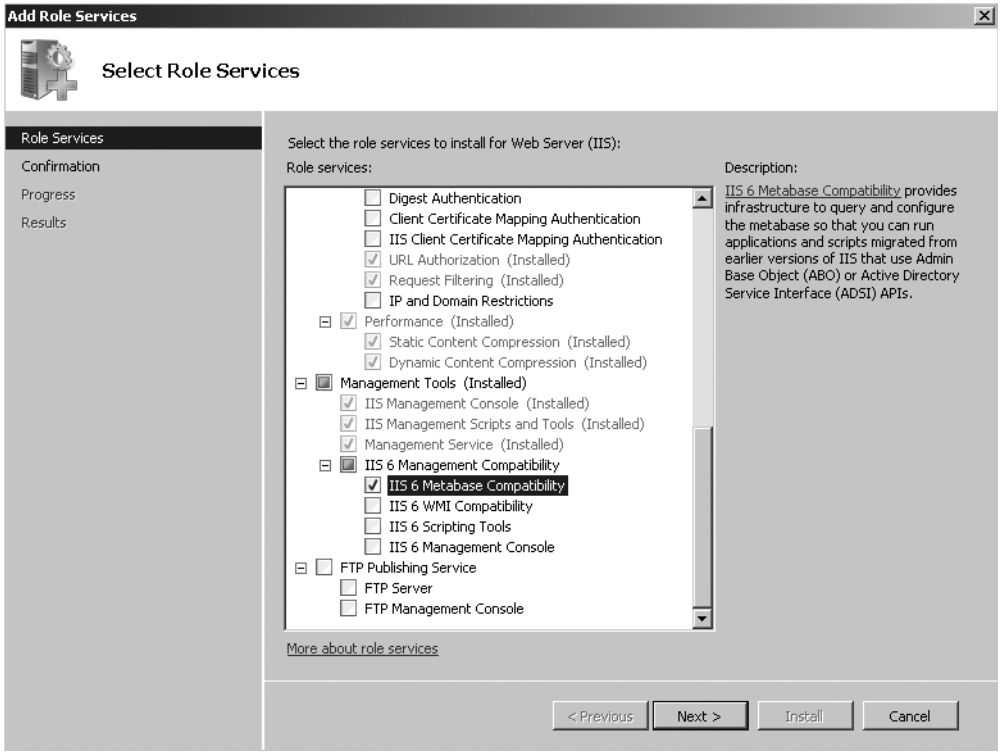


Figure 4-2 Installing IIS 6.0 Metabase Compatibility with Server Manager.

You can also chose to install the legacy IIS 6.0 configuration scripts from the IIS 6.0 Metabase Compatibility category, which provides scripts such as adsutil.vbs and iisweb.vbs. However, we recommend that for your configuration scripts and programs, you start to use the new configuration tools and APIs that the IIS 7.0 configuration system provides.

IIS 7.0 and the .NET Configuration Systems

The .NET configuration files on IIS 7.0 (machine.config, root web.config, and application web.config) behave exactly the same as they do on IIS 6.0. In fact, the .NET configuration system isn't really aware it's running on IIS 7.0 and does not read any of the IIS 7.0 configuration settings. However, IIS 7.0 is very aware of .NET. The IIS 7.0 configuration hierarchy includes the server-level .NET configuration files, machine.config and root web.config (in addition to applicationHost.config), but the .NET configuration system does not include IIS configuration stored in applicationHost.config.

One of the primary benefits of this design is that IIS 7.0 configuration settings can be stored in the same distributed web.config configuration files as the ASP.NET configuration settings. This enables applications to contain all of the configuration they need to run on the IIS platform in the web.config file, and it also enables simple xcopy deployment.

From a developer perspective, it also enables managed modules developed for IIS 7.0 to access .NET configuration by using IIS 7.0 Microsoft.Web.Administration and other .NET classes in the same way they can access IIS 7.0 configuration sections. Likewise, the IIS 7.0 configuration APIs can be used to manage the .NET configuration sections in automated deployment and management scenarios.

In addition, the IIS Manager tool exposes a number of ASP.NET configuration features. For example, you can configure database connection strings in the IIS Manager instead of having to open up the .config file. IIS Manager also enables you to manage users and roles by using the .NET role and membership providers. This is very useful for managing user information for features such as forms authentication and storing IIS Manager users. You can learn more about IIS Manager support for ASP.NET features in Chapter 6, "Using IIS Manager."

The unification of the .NET and IIS 7.0 configuration hierarchies does pose a few issues that stem from the fact that the two configuration systems have completely separate implementations, yet they work with the same configuration hierarchy and configuration sections. The fact that the ASP.NET configuration system does not read IIS 7.0 configuration sections eliminates a lot of potential problems with the differences in behavior. However, some problems do still exist.

One of the key limitations stems from the difference in encryption support between the two configuration systems. The .NET configuration files may contain user names and passwords that the developer can encrypt. This way, when you view the .config file, you see an encrypted secret rather than plain text. The problem arises because IIS 7.0 and the .NET configuration system use different methods for encrypting secrets. The .NET configuration system supports section-level encryption, which encrypts the entire contents of the configuration section. The IIS 7.0 configuration system supports only attribute-level encryption, which encrypts specific attributes. Because of this, if you attempt to read an encrypted ASP.NET configuration section through the IIS 7.0 configuration system or any of the APIs that use it, you will receive an error. For example, this will happen if you encrypt any of the configuration sections that the IIS Manager uses to administer ASP.NET functionality. Likewise, you cannot encrypt ASP.NET configuration sections with IIS 7.0 configuration encryption because ASP.NET will fail to read their contents. For more details on this issue and how to solve it, see Chapter 14. Another limitation stems from the lack of a versioning mechanism for the .NET configuration schema files provided by the IIS 7.0 configuration system. As of this writing, the IIS 7.0 configuration system provides schema files only for the .NET Framework 2.0 configuration, and therefore IIS 7.0 might experience problems when writing configuration to configuration files for .NET Framework 1.1 or future versions of the .NET Framework. Moreover, some of the tools in the IIS 7.0 configuration stack, including Appcmd.exe, can't write to .NET Framework configuration files for versions other then 2.0. Future versions of IIS may address this problem.

The use of IIS 7.0 configuration in ASP.NET web.config files may also create a problem for ASP.NET applications that are using .NET Framework 1.1. This is because the ASP.NET configuration system is not aware of the IIS 7.0 configuration sections located in the

<system.webServer> section group (or any custom configuration sections you create using the IIS 7.0 section extensibility mechanism), and the configuration system will generate an error when it encounters these sections in web.config files. ASP.NET 2.0 includes a special configuration declaration in machine.config that maps <system.webServer> to a special configuration section handler type that ignores the sections when they are found.

```
<section name="system.webServer" type="System.Configuration.IgnoreSection,
System.Configuration, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />
```

However, ASP.NET 1.1 does not include this special configuration declaration because ASP.NET was released long before IIS 7.0 development began. Therefore, you may need to manually add this section declaration for <system.webServer> and other custom IIS 7.0 sections/section groups so that you can use them in web.config files.

Editing Configuration

The IIS 7.0 configuration system provides a lot of flexibility for editing server configuration. Because the configuration is stored in plain-text XML files and uses a well-structured, human-readable syntax, you can edit it manually using Notepad or your favorite text editor. In fact, many people prefer this approach when editing configuration for distributed web.config files located within the site's directory structure.

In addition to enabling configuration to be edited by hand, IIS 7.0 provides a complete administration stack that offers tools and APIs for editing configuration. This includes the IIS Manager, a completely redesigned GUI task-based experience for managing most of the IIS 7.0 configuration. It also includes the Appcmd command line tool, which you can use to edit configuration quickly from the command line. Finally, it includes several administrative scripts and APIs for editing configuration programmatically, including the IIS 7.0 configuration COM objects that can be accessed from native C++ programs (called the Application Host Administration objects or AHADMIN) and Windows scripts, a WMI provider, and new Microsoft.Web.Administration APIs for managing configuration from .NET programs.

Note Where possible, use tools to manipulate IIS 7.0 configuration instead of changing configuration by hand. This is much easier and provides protection against generating incorrect configuration.

In fact, you should choose to use tools to edit the configuration on the server, because doing so ensures that you are interacting correctly with the underlying complexity of the configuration system and guarantees that the configuration is written using the correct syntax. The IIS Manager is a great way to do this, because it provides a simplified task-based view of many IIS 7.0 features, so you don't need to understand their configuration structure. You can read about managing IIS 7.0 with the IIS Manager in Chapter 6.

However, there are times when you need to specify configuration by hand or use one of the lower-level tools like Appcmd or programmatic interfaces like the Microsoft.Web.Administration namespace provided in .NET. In this case, you do need to understand the structure of configuration sections and inheritance behavior of the configuration hierarchy in order to do this correctly. In the remainder of this section, we will discuss the basics of editing IIS 7.0 configuration that will help you to do it correctly.

Note Use Appcmd to edit configuration in situations in which IIS Manager does not expose the desired configuration functionality. Appcmd can perform most configuration tasks you can do by hand, and it offers the benefit of additional validation. It also allows you to perform configuration tasks in an automated fashion on other machines if needed. For more information on using Appcmd, see Chapter 7, "Using Command Line Tools."

Caution Before modifying configuration, always make sure you have a backup of the current state so you can come back to it if necessary. See the section titled "Backing Up Configuration" later in this chapter for more information on how to easily back up and restore IIS configuration.

Deciding Where to Place Configuration

Earlier in the chapter, I described the IIS 7.0 configuration hierarchy. This hierarchy contains multiple configuration files, comprising the .NET configuration files, applicationHost.config, and distributed web.config files in your site directory structure. This hierarchy allows you to map configuration to a URL namespace on your server by placing it in the right configuration file. When the server reads configuration for a particular Web site or URL, it merges all configuration files along the configuration path, merging the configuration specified in them to achieve the effective set of configuration for a given path.

Because of the configuration merging, configuration specified at a higher configuration path always inherits to all child paths, unless it is overridden lower down. For example, configuration specified in applicationHost.config is inherited by all sites and URLs on the server, unless it is overridden in their respective web.config files.

Table 4-5 indicates where you may chose to place configuration in order to apply it to the desired scope.

Table 4-5 Placement of Configuration

Configuration For	Place In
Entire server	applicationHost.config
A specific site	web.config in the site's physical root directory
A specific application	web.config in the application's physical root directory
A specific virtual directory	web.config in the virtual directory's physical root
A specific URL	If the URL corresponds to a physical directory, in web.config in that directory; otherwise, in any existing parent web.config file with a *location tag* for the specific URL

When specifying configuration at a specific site or URL, you always have a choice of specifying configuration in a distributed web.config file corresponding to the URL or placing it in a configuration file higher in the hierarchy (for example, applicationHost.config) and applying it to the specific URL by using location tags. Both have advantages and disadvantages you need to consider.

Using location tags can allow you to place all configuration in a single location, instead of multiple web.config configuration files which may be harder to discover and manage. Also, if configuration is locked at a particular configuration path (for example, configuration that should only be set by server administrators is typically locked in applicationHost.config), you are forced to use location tags at that path in order to apply configuration to child paths. However, placing configuration in distributed web.config files allows the site/application/directory to become portable and xcopy-deployed to other servers or places in the site structure without having to set any configuration elsewhere or requiring administrative privileges on the server. This is a very powerful ability.

Finally, a note about configuration delegation—not all configuration sections are allowed to be specified in distributed web.config files by default. It is up to the server administrator to decide which configuration sections are delegated and to unlock them in applicationHost.config. This may impact your ability to run applications that specify configuration in distributed web.config files, generating errors if locked configuration is specified. We will discuss managing configuration delegation in the section titled "Delegating Configuration" later in this chapter.

Setting Configuration

To set configuration, you need to know three things: the name of the section that contains the desired configuration settings, the desired property of that section, and the configuration path at which you want to set this setting to apply (as we discussed in the previous section). You will typically know the first two from the documentation of the feature you are attempting to configure. For more information about what configuration sections are available and their format, you can consult the schema files in the %windir%\system32\inetsrv\config\schema directory.

When you know this information, you can specify the corresponding section element in the configuration file.

```
<configuration>
  <system.webServer>
    <defaultDocument … />
  </system.webServer>
<configuration>
```

Note the <configuration> element—this must always be the root element of any configuration file. Also, notice the <system.webServer> element—this is the section group element for the <defaultDocument> section (and all other IIS 7.0 configuration settings) that is being configured.

Configuration sections contain the properties that you intend to configure, such as *defaultDocument*, but you need to do more than just provide a name. You turn the default document feature on and off and provide the list of default documents using attributes or collection elements contained inside the section.

Setting Section Attributes

The majority of configuration settings are expressed via attributes, which may either be exposed on the collection element itself or in one of the child elements of the collection.

To specify a value for the attribute, you simply need to set the value of that attribute. This effectively overrides any default value or value previously set to this attribute in earlier configuration paths. Following is an example of setting the *enabled* value on the <defaultDocument> section.

```
<defaultDocument enabled="true" />
```

Each attribute has a specific type and may have additional validation rules associated with it in the schema definition of the section. Likewise, attributes may be given default values that are taken on by them if they are not explicitly set in configuration. This will be documented for each section to assist you in setting their values.

Manipulating Configuration Collections

In addition to attributes, configuration sections can also contain collections. Collections allow lists of items to be represented in configuration, and they support additional behaviors such as adding or removing elements in multiple configuration levels and preventing duplicate items from being added.

Collections are typically configured through three different operations: adding collection elements, removing collection elements, and clearing the collection.

Adding Items to a Collection with <add /> To add items to a collection, you typically use the <add /> element and specify the desired attribute values inside of it. For example, following is an excerpt from the <files> collection of the <defaultDocument> section specified in applicationHost.config after installation.

```
<defaultDocument enabled="true">
    <files>
        <add value="Default.htm" />
        <add value="Default.asp" />
        …
    </files>
</defaultDocument>
```

In this case, elements in the <files> collection only support a single attribute called “value”. However, collection elements are not limited to a single attribute—they can define any number

of attributes, child elements, or even subcollections. In fact, each collection element has the same schema flexibility as any other configuration element or the section itself. Following is an example from the <sites> section.

```
        <sites>
            <site name="Default Web Site" id="1">
                <application path="/">
                    <virtualDirectory path="/
" physicalPath="%SystemDrive%\inetpub\wwwroot" />
                </application>
                <bindings>
                    <binding protocol="http" bindingInformation="*:80:" />
                </bindings>
                <traceFailedRequestsLogging enabled="true" />
            </site>
        </sites>
```

The <sites> section is a collection of <site> elements (notice that it uses <site> as the name for its <add> element—this is a capability provided by the IIS configuration schema that some sections take advantage of for readability). Each <site> element in turn is a collection of <application> elements, which in turn contain a collection of <virtualDirectory> elements. Each <site> element also has a <bindings> child element, which itself is a collection of site bindings. You can find a detailed description of the new site, application, and virtual directory structure in Chapter 9, "Managing Web Sites."

Luckily, the <sites> section is the most complicated section on the entire IIS 7.0 configuration schema, and most other sections are a lot simpler.

Most collections enforce item uniqueness to prevent duplicate items from being added. This is done by marking one or more of the attributes allowed on the collection <add> elements as the collection key. If an item with a duplicate key is specified, the collection will trigger a configuration error when accessed.

When you add collection elements at a particular configuration level, they add to the existing elements that were inherited from a parent level. For example, the <defaultDocument> section can use this to specify a base set of default documents in applicationHost.config and then add specific default documents at the site or virtual directory levels.

The ordering of collection items inside a collection is determined by the order in which they are added. When collection items are inherited from the parent configuration levels, they are placed before the collection items specified at the current level. This is true for most collections, except for collections that elect to have a prepend order—these collections place the elements declared at the current level before elements inherited from parent levels. These include the IIS <handlers> and the ASP.NET <authorization> sections.

Removing Items from a Collection with <remove /> Because of the collection inheritance, it is sometimes necessary to remove elements that are declared at a higher configuration level. For example, you may want to remove a specific module from the <modules> configuration

collection for a specific application if you do not need this module to run. For more about managing modules, see Chapter 12.

Note If you are removing a collection element that is added at the current configuration level, you can simply delete the corresponding <add> element. Use <remove> to remove the elements that are specified by parent configuration levels.

To do this, you can use the <remove> element. Each remove element specifies the attributes that together comprise the collection key to uniquely identify the element that is to be removed. For example, following is the configuration you can use to remove "Default.asp" from the <files> collection of the <defaultDocument> section.

```
<defaultDocument>
    <files>
        <remove value="Default.asp" />
    </files>
</defaultDocument>
```

Clearing the Collection with <clear /> Sometimes you may want to completely clear the collection items that are defined by the parent configuration levels and specify only the items that are required. This is often done whenever the current configuration level has to have complete control over the contents of the collection and cannot inherit parent items.

This is accomplished with the <clear/> element. The <clear/> element removes all of the inherited collection items, leaving only the items that are added at the current level after the <clear/> element. The following example clears the default document collection and adds back a single element to make sure that only Default.aspx is treated as a default document.

```
<defaultDocument>
    <files>
        <clear/>
        <add value="Default.aspx" />
    </files>
</defaultDocument>
```

Important Be careful when using the <clear/> element, however, because it completely stops the inheritance of parent collection items to the current configuration level or its children. This means that if the administrator adds new collection items at the server level, they will not be propagated to the current level. Therefore, use <clear/> only when you want to take complete control over the contents of the collection.

Understanding Configuration Errors

In contrast to IIS 6.0, when editing configuration with tools like the IIS Manager and Appcmd, or programmatically with APIs like Microsoft.Web.Administration, the underlying configuration system APIs will make sure that the resulting configuration is correct. This will

catch most attempts to produce incorrect configuration, including using data of the wrong type for attribute values, attempting to set nonexisting attributes, or using data out of range of accepted values. It will even prevent you from adding a duplicate collection element or attempting to write configuration that has been locked at a parent configuration level. This is the reason why you should always prefer to use tools to write configuration, rather than doing it manually.

Note Use tools to set configuration—this will catch most mistakes and prevent you from generating incorrect configuration.

However, there are times when you may still run into a situation in which configuration is incorrect. This is most likely if you edit configuration by hand and make a mistake in the section syntax or set attributes to unsupported values. However, it may also happen in other cases—for example, if an application that defines configuration is deployed to a server where some of the sections are locked at the server level, resulting in a lock violation.

Because of this, it is important to be able to understand various configuration error conditions, and be able to use the resulting configuration error information to resolve them.

Caution Always back up configuration before making changes to it. You can learn more about backing up configuration in the section titled "Backing Up Configuration" later in this chapter.

There are several types of configuration errors that are handled differently by the configuration system and have varying degrees of impact on IIS. Table 4-6 summarizes some of the common error conditions and the impact they have on the server.

Table 4-6 Common Error Conditions

Error	Impact
Configuration file is not valid XML	■ If Framework machine.config, root web.config, or IIS 7.0's applicationHost.config: the entire server will be taken offline. ■ Otherwise: All URLs corresponding to the configuration file and below will return configuration errors.
Configuration file cannot be accessed: The file is locked by another process, access denied, no network connectivity for UNC paths.	■ If Framework machine.config, root web.config, or applicationHost.config: the entire server will be taken offline. ■ Otherwise: All URLs corresponding to the configuration file and below will return configuration errors.

Table 4-6 Common Error Conditions

Error	Impact
Configuration section syntax error: The configuration section has unexpected elements or attributes, or it is missing required attributes.	■ If the error is in one of the system.applicationHost configuration sections that are read by WPAS, the server may be taken offline. ■ If the error is in one of the core Web server sections, all requests to the URLs affected by the errors will return configuration errors. ■ Otherwise, requests that use features that read the configuration section will return configuration errors.
Attribute validation error: There is an invalid data type; value fails attribute validation rules.	Same as above.
Collection validation error: There are duplicate collection elements.	Same as above.
Lock violation: Specifying configuration for the section or attribute that is locked at a parent level.	Same as above.

The key to understanding these error conditions is to understand how the configuration system handles errors. Errors that cause the entire configuration file to become unavailable, because it cannot be read or because it contains invalid XML (as shown in Figure 4-3), cause all attempts to read configuration from that file to fail. Because of this, all operations that require reading this file will fail—if this file is applicationHost.config, which is read by the Windows Process Activation Service component of IIS that is responsible for managing IIS worker processes, the entire server will be taken offline. In this case, you will not be able to get a detailed request error describing the error condition, because the server will not be able to start any IIS worker processes to serve the request. In this case, the error information will be logged by WPAS to the System EventLog.

If the file is a distributed web.config file that corresponds to a particular URL namespace, that namespace will not be available. However, IIS worker processes will still be able to start and generate a detailed configuration request error that will describe the reason, and sometimes even the position in the file, where the error has occurred.

Finally, for all other errors in configuration sections that are not invalid XML, only accesses to the affected section will fail. If the error is in one of the system.applicationHost sections that are read by WPAS, including <sites> and <applicationPools>, WPAS may again fail to start IIS worker processes, resulting in the entire server being offline and errors being logged to the System EventLog. If the error is in one of the core IIS configuration sections that are read on every request, which include <serverRuntime>, <modules>, and <handlers>, all requests to the URL namespace corresponding to the invalid configuration will return configuration errors.

These errors will contain the exact reason why the configuration access failed, including details such as the line number and the element or attribute in question that has incorrect configuration, as shown in Figure 4-4. You can use this information to quickly pinpoint the location of configuration syntax error and resolve it.

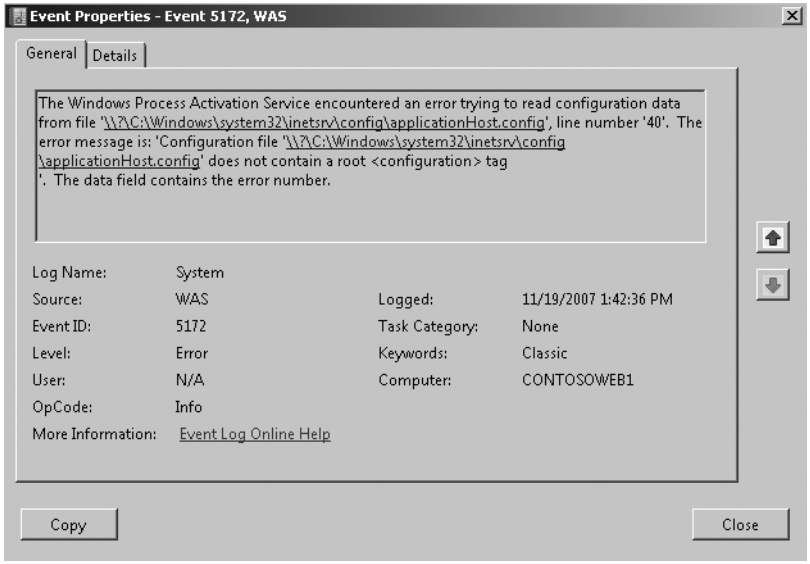


Figure 4-3 EventLog error from malformed XML in applicationHost.config.

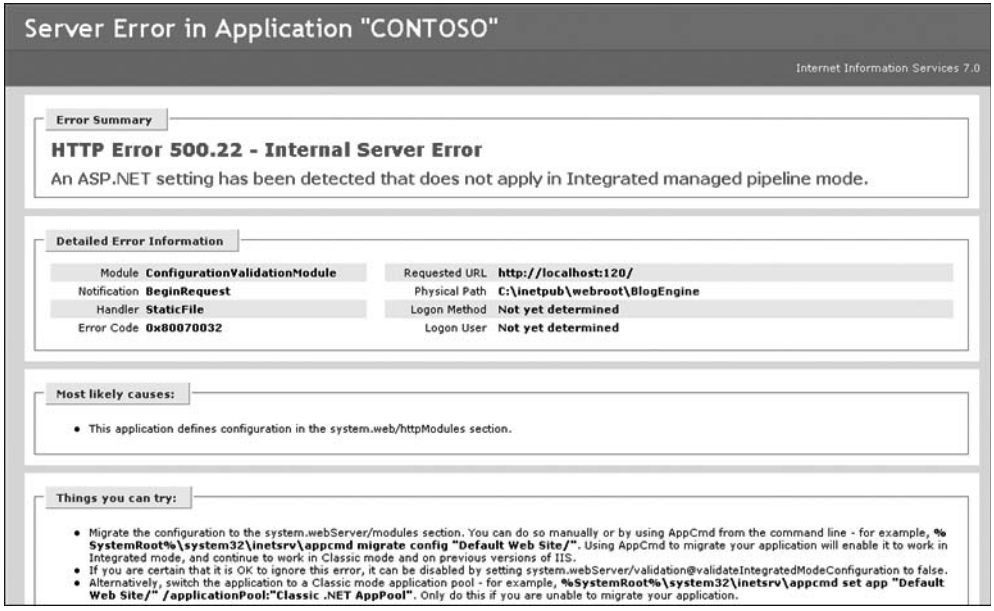


Figure 4-4 IIS 7.0 configuration error message.

Note To see the detailed configuration error, you will need to either make a request locally on the server or enable detailed errors.

For all other sections, only requests that use features whose configuration has the error will trigger request errors. This also means that if you make a mistake in configuration for a feature that is not being used (for example, the module is disabled), no error will be given and invalid configuration will remain ignored.

Finally, if the error is in an ASP.NET configuration section, which is read by the ASP.NET using the .NET configuration system, you may get an ASP.NET exception error page containing the configuration error details.

Note To see the detailed ASP.NET configuration exception, you will need to either make a request locally on the server or enable ASP.NET detailed errors.

Managing Configuration

In the course of working with IIS configuration, you will need to perform a variety of management tasks in addition to editing the configuration itself. Notably, you will need to back up and restore configuration, in order to revert from unintended changes or recover from corrupted configuration files. This is especially critical because the ease of editing IIS XML configuration files also makes it easy to make undesired changes.

In fact, when working with IIS configuration, you should always insure that you make a backup that can be used to go back to the state before the changes. Luckily, IIS makes it very easy to do this.

In this section, we will review the management tasks around backing up and restoring IIS configuration. We will also discuss setting up shared configuration between multiple servers and setting up configuration delegation that enables some configuration to be set in distributed web.config configuration files.

Backing Up Configuration

Before making changes to IIS configuration files, you should back them up so that you can restore them later if your changes corrupt configuration or result in incorrect server operation. The latter is a critical reason—the server may look like its working properly initially until a future time when problems are detected, at which point you may want to come back to the previous configuration state.

Typically, it is not necessary to make special arrangements to back up delegated configuration located in web.config inside your Web site structure, because those files are backed up

together with your site content (of course, you need to maintain backups of your site content for this to work).

However, if you make changes to the server-level configuration files, you should make a backup of server configuration. Thankfully, IIS 7.0 makes it easy to do that via the Appcmd command line tool.

From an administrative command prompt, type

%windir%\system32\inetsrv\ AppCmd Add Backup MyBackup

This creates a backup of IIS configuration files, including applicationHost.config, redirection.config, and administration.config, and custom schema files if there are any. The backup is created as a named directory under the %windir%\system32\inetsrv\backup directory, using the name you specified to the "Add Backup" command. This directory will contain the backed-up files.

Note If you do not specify a backup name, Appcmd will automatically create a name using the current date and time.

You can list the backups made on your system by using the List Backups command.

```
%windir%\system32\inetsrv\AppCmd List Backups
```

Then, you can restore any of the listed backups by using the Restore Backup command.

```
%windir%\system32\inetsrv\AppCmd Restore Backup "MyBackup"
```

The restore command will restore all of the files in the backup folder, overwriting the current server configuration with those files. No confirmation prompt is given, so always consider backing up the current configuration first before restoring another set.

A note about configuration file security and encryption: the backup process simply copies the server configuration files to the inetsrv\backup directory, which by default is secured with the same NTFS permissions as the inetsrv\config directory, which contains the original files. If the files contain encrypted configuration, those details will stay encrypted in the backed-up copies. No additional encryption is performed as part of the backup mechanism. Therefore, the files are only protected when they are in the backup directory and are not safe to place in an offline location without additional protection.

Using Configuration History

By default, IIS 7.0 via the AppHostSvc will check every two minutes to see if application-Host.config has changed, and if so will make a backup of the file. You'll find the backed-up configuration files in the Inetpub\history folder by default. You can change both the location

of the backups as well as several other configurable parameters in the <configHistory> configuration section, as shown in Table 4-7.

Table 4-7 <configHistory> Attributes

Attribute	Default Setting	Definition
Enabled	True	This value indicates whether configuration history is enabled or disabled
Path	%systemdrive%\inetpub\history	The path where history directories will be created and stored
maxHistories	10	The maximum number of directories retained by IIS 7.0
Period	00:02:00	The time between each check made for changes by IIS 7.0

If you do nothing at all, the values listed in Table 4-7 are preconfigured for you. To modify these values, you need to enter them into applicationHost.config, because the IIS Manager does not have a UI for configuring this section of applicationHost.config. You can use Appcmd for this. For example, the following command will change the path for storing backups to %systemdrive%\MyWebHistory. Note that the path must exist first or the service will not work.

```
%windir%\system32\inetsrv\Appcmd set config /section:configHistory
"/path:%systemdrive%\MyWebHistory"
```

You can use the **Appcmd Restore Backup** command to restore any of the configuration history backups the same way you restore manual backups performed by the **Appcmd Add Backup** command. You can list all of the available backups, including both manual and configuration history backups, by doing the following.

```
%windir%\system32\inetsrv\AppCmd List Backups
```

For more information about configHistory, see the article "Using IIS7 Configuration History" at *http://www.iis.net/articles/view.aspx/IIS7/Managing-IIS7/Configuring-the-IIS7-Runtime/Understanding-AppHost-Service/Using-IIS7-Configuration-History?Page=1*.

Exporting and Importing Configuration

By default, IIS 7.0 configuration stores no secrets and therefore is not tied to a specific server as it was in previous versions. The reason for the IIS 6.0 metabase to be tied to a local server and protected is that by default it contains the passwords for the anonymous user and IWAM user. If these passwords were discovered, it is feasible they could be used to log on to the server. They were random and complex, which provided a high very high degree of security.

In IIS 7.0, the anonymous user (IUSR) is a "built-in" account rather than a local account, so it does not require a password. Don't worry, even though there is no password, you can't use this built-in account to log on to the server. There is no possibility that the IUSR account can

be used to log on locally or remotely except through IIS. In addition, there is no IWAM account, since IIS5 application isolation mode is not part of IIS 7.0. Since there are no secrets by default in applicationHost.config, there is no need to key it to an individual server.

This means that you can take applicationHost.config from one server and copy it to another server provided you also synchronize the server encryption keys, presuming the target server has the same content and directory structure. This provides a simple mechanism for exporting and importing configuration between servers.

Note To use the applicationHost.config file from one server on another server, you do need to make sure the servers use the same configuration encryption keys. This is because applicationHost.config contains encryption session keys that are themselves encrypted using the server's RSA configuration key. You can learn more about exporting and importing server encryption keys in the section titled "Sharing Configuration Between Servers" later in this chapter.

In the case in which your configuration files do contain encrypted information, such as application pool identities, the configuration files are tied to the specific server on which the encryption information is generated. You can, however, export and import the configuration keys in order to allow multiple servers to share the same encrypted configuration—in fact, this is one of the requirements for the shared configuration feature supported by IIS 7.0. You can learn more about setting up shared configuration later in this chapter. You can also find an in-depth discussion of configuration encryption in Chapter 14.

Unlike IIS 6.0, IIS 7.0 does not provide a built-in mechanism to export configuration for a particular site, as opposed to exporting the entire server's configuration. In a lot of cases, this can be accomplished by manually re-creating the site definition on the target server and then simply copying the site content, which can now define its configuration in the web.config files contained within the site's directory structure.

However, if the site configuration is located inside location tags in applicationHost.config, there is no automated mechanism to export it. You can, of course, simply copy the contents contained in the location tag (including the location tags) and add it to the bottom of another applicationHost.config. An automated mechanism may become available in the future.

Delegating Configuration

The new configuration system in IIS 7.0 was designed to provide rich support for feature delegation. This term has a special meaning in IIS 7.0—the ability to designate features that Web site administrators or application managers can control at the site or application level—without making them administrators on the server. As you will see, feature delegation works hand in hand with remote administration and is built into the IIS Manager, which allows you to configure delegation and at the same time respects delegation settings, limiting access to locked or limited features.

Feature delegation is implemented in two ways. First, the configuration hierarchy itself allows configuration to be specified in distributed web.config files, which are typically under control of the site administrator or application developer who do not have to be server administrators to set or change configuration therein. The server administrator has control over what configuration can be set in the delegated manner in web.config files, versus what configuration can only be set by a server administrator in applicationHost.config. This control is accomplished through configuration locking, which can be done at the section level by locking the section in applicationHost.config or at the granular level by locking specific configuration settings in a particular configuration section. Granular configuration locking is described in more detail in this chapter in the section titled "Granular Configuration Locking."

The second way is implemented by IIS Manager, which subsumes the configuration section locking mechanism and provides a way to manage the delegation of the underlying configuration and the corresponding IIS Manager UI features for seamless integration with remote administration through the tool. Managing feature delegation through the IIS Manager has the advantage of ensuring correctly configured delegation. The IIS Manager will respect delegation settings so that a remote user cannot see features that are hidden (marked as Not Delegated in the IIS Manager), and cannot make changes to features that are marked as Read Only in the IIS Manager.

Important Any user that can upload a web.config can overwrite IIS 7.0 and ASP.NET settings in web.config. If you use the IIS Manager to write configuration, these settings will be properly maintained and users will only be allowed to change configuration for which they have access. If a web.config file is created outside of using the IIS Manager and then uploaded to the site, it may contain configuration settings that are not permitted by the delegation settings. In this event, IIS 7.0 will present a configuration locking error, and the previous, correct, web.config details may be lost, since the original web.config has been overwritten.

When you delegate control to others, there will be a strong incentive for them to control their site or application configuration using the IIS Manager, as it will show only features that the user has the right to see or control.

In general, features in IIS 7.0 are related to configuration "sections" in applicationHost.config. We've already described this in the discussion earlier in this chapter on section definitions and the value for "overrideModeDefault" associated with each section. The IIS Manager, of course, is the main tool for controlling configuration of these sections, and it's much easier to understand and manage delegation using the IIS Manager than any other way.

Delegation Settings in the IIS Manager

Let's examine the various settings in the IIS Manager related to delegation. Figure 4-5 shows the results you'll see if you select the server node in the tree view and then Feature Delegation from the features pane.

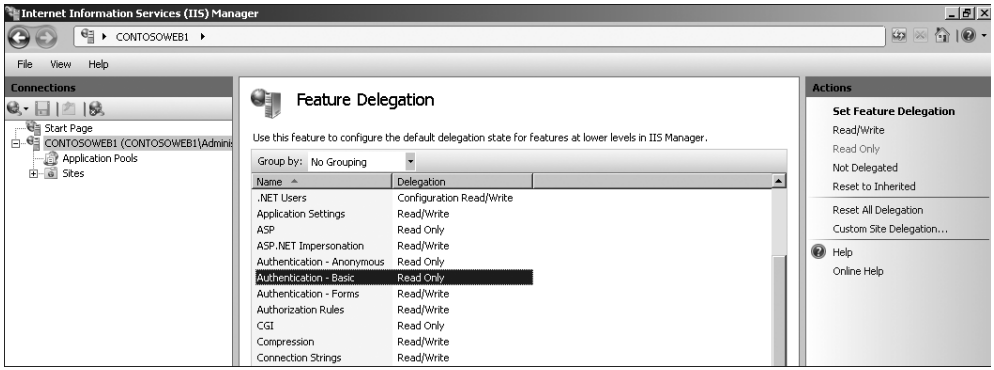


Figure 4-5 Feature Delegation in the IIS Manager.

The Delegation column lists the current delegation setting for each feature. The names for these various states for delegation may not be as clear as they might be to describe what's going on, so you should not try to infer a great deal from the terms:

- **Not Delegated** When a feature is marked as Not Delegated, the corresponding configuration section will be locked in applicationHost.config by placing it inside a <location> tag with the **overrideMode** value set to Deny. When a feature is marked as Not Delegated, any changes you make to this feature at server level (that is, with the server icon selected in the tree view) will be recorded in applicationHost.config. Changes at the site or application level can only be made by the server administrator and will be recorded in applicationHost.config using <location> tags to apply them to the required path. When using the IIS Manager to connect to the site or application, remote users will not be able to see the corresponding feature icon or change its settings. If a web.config file is uploaded that contains settings for a Not Delegated feature, a configuration error occurs.
- **Read Only** This is the same as Not Delegated, except that remote users will be able to see this feature; however, they cannot change any values. This is a useful setting when you want users to know, for example, what authentication methods are available to them, but you don't want them to be able to turn them on or off.
- **Read/Write** When a feature is marked as Read/Write, the configuration section will be unlocked for distributed web.config files. This is accomplished by placing the configuration section in a <location> tag with the overrideMode value set to Allow. Any changes you make to this at feature at server level will be recorded in applicationHost.config. Changes to this feature at the site or application level will be recorded in the appropriate web.config. (A reference to site level designates the web.config in the site root. The application level refers to the web.config file that resides in a folder within the site that has been designated as an application.) When using the IIS Manager to connect to the site or application, remote users will be able to see and change the settings.

Additional delegation values may be provided by third-party extensions to the tool that have extension-specific meaning.

You can learn more about configuring IIS Manager feature delegation and determining which users have the right to manage the Web server configuration remotely in Chapter 8, "Remote Administration."

Default Settings for Delegated Configuration

As mentioned, certain settings in IIS 7.0 are delegated by default, whereas others are specifically locked down. Table 4-8 is from a prerelease version of the *IIS 7.0 Hosting Deployment Guide*, which can be located on IIS.net. The information in the table details which features are delegated and why. You may want to make different decisions than the IIS team regarding these default settings, but a great deal of thought has gone into these settings, so we would advise not making changes to the global settings without good reason.

Table 4-8 Features and Delegated Settings

Feature	Delegated Setting	Reason
.NET Compilation	Read Only (changed from Read/Write)	Specifies settings for ASP.NET compilation processing directives like the temporary compilation directory. Prevents users from setting the temporary compilation directory manually.
.NET Globalization	Read/Write	Specifies settings for default culture and globalization properties for Web requests.
.NET Profile	Read/Write	Specifies settings for user-selected options in ASP.NET applications.
.NET Roles	Read/Write	Specifies settings for groups for use with .NET users and forms authentication.
.NET Trust Levels	Read Only (changed from Read/Write)	Specifies the trust level. By locking down the trust level when you follow the ASP.NET guidance in this document, you will be setting this to Read Only and locking it for the server. Prevents Web site owners from setting the trust level to a higher level than set by the server administrator. For example, if a custom trust level is set by the administrator, this setting should be set to Read Only so it cannot be overridden.
.Net Users	Configuration Read/Write	Specifies settings for management of users who belong to roles and use forms authentication.
Application Settings	Read/Write	Specifies settings for storing data (name and value pairs) that managed code applications can use at run time.
ASP	Read Only	Specifies Classic ASP settings.
ASP.NET Impersonation	Read/Write	Specifies impersonation settings. Site owners can use this to run their site under a different security context.

Table 4-8 Features and Delegated Settings

Feature	Delegated Setting	Reason
Authentication—Anonymous	Read Only	Specifies anonymous authentication settings.
Authentication—Forms	Read/Write	Specifies forms authentication settings.
Authentication—Windows	Read Only	Specifies Windows authentication settings.
Authorization Rules	Read/Write	Specifies the list of Allow or Deny rules that control access to content.
CGI	Read Only	Specifies properties for CGI applications. Should be left set to Read Only to prevent users from changing settings.
Compression	Read/Write	Specifies settings to configure compression.
Connection Strings	Read/Write	Specifies connection strings that applications can use.
Default Document	Read/Write	Specifies default documents for the Web site. By leaving this Read/Write, users will be able to specify a custom default document for their site without contacting the server administrator.
Directory Browsing	Read/Write	Specifies directory browsing settings.
Error Pages	Read Only	Specifies what HTTP error responses are returned.
Failed Request Tracing Rules	Read/Write	Specifies settings for failed request tracing rules. Enables users to create rules for tracing requests based on parameters like time taken or status code and to diagnose problems with their site.
Feature Delegation	Remove Delegation (changed from Read/Write)	Specifies settings for delegating features to applications. It can be turned off unless server administrators want to enable this feature for site owners.
Handler Mappings	Read/Write	
HTTP Response Headers	Read/Write	Specifies HTTP headers that are added to responses from the Web server.
ISAPI Filters	Read Only	Specifies ISAPI filters that process requests made to the site or server, such as ASP.NET.
Logging	Remove Delegation	
Machine Key	Read/Write	Specifies hashing and encryption settings for applications services, such as view state, forms authentication, and membership and roles.
MIME Types	Read Only	Specifies what file types can be served as static files.

Table 4-8 Features and Delegated Settings

Feature	Delegated Setting	Reason
Modules	Read/Write	Specifies native and managed code modules that process requests made to the site or server.
Output Caching	Read/Write	Specifies rules for caching output.
Pages and Controls	Read/Write	Specifies page and control settings for applications.
Redirect Rules	Read/Write	Specifies settings for redirecting requests to another file or URL.
Session State	Read/Write	Specifies session state and forms authentication cookie settings.
SMTP E-mail	Read/Write	Specifies e-mail address and delivery options for e-mail sent from the site.
SSL Settings	Read Only	Specifies settings for SSL.

Directly Configuring Delegation

Although you can manage the delegation of many IIS features in the IIS Manager, it only allows you to manage the underlying configuration delegation for features that have corresponding UI pages in the IIS Manager. For those features, selecting the IIS Manager delegation state also generates the required configuration delegation settings to control whether the corresponding configuration sections can be used at the site or application level.

However, there are times when you will need to manage configuration delegation directly. One such case is when the configuration section does not have a corresponding IIS Manager feature. For example, IIS 7.0's URL Filtering feature does not, at the time of this writing, have a UI component. In these cases, you can work with the configuration system directly or the Appcmd command line tool to configure the desired configuration delegation.

The initial ability to delegate a specific configuration section is controlled by the **overrideModeDefault** attribute on its declaration (see the "Section Declarations" section earlier in this chapter). Some of the built-in IIS 7.0 configuration sections like <defaultDocument> allow delegation by default by specifying Allow for this attribute in their declarations, and others like <serverRuntime> do not by specifying Deny. This decision is typically made by the developer of the feature that reads this configuration section, based on whether or not the feature configuration should be by default delegated to users who are not server administrators.

Caution Do *not* change the **overrideModeDefault** setting on section declarations to unlock them. The IIS team recommendations for default delegation settings are well reasoned. If you need to override the default setting globally, use Location tags referencing the "*" path (or a null path, "").

The overrideModeDefault setting on the section declarations in applicationHost.config sets the default value for delegation. You can modify the delegation status of each configuration section by locking or unlocking it. Unlocking sections is often needed in order to be able to specify configuration for certain sections in web.config files of your Web site. Likewise, you may want to lock certain other sections if you do not want the Web sites on your server to be able to override the settings set in applicationHost.config.

To unlock a section, you can use the Appcmd.exe command line tool as follows.

```
%windir%\system32\inetsrv\AppCmd Unlock Config /section:<SectionName>
```

Where <SectionName> is the name of the section, for example, "system.webServer/serverRuntime".

To lock a section that is currently unlocked, you can use the following command.

```
%windir%\system32\inetsrv\AppCmd Lock Config /section:<SectionName>
```

Locking or unlocking a section produces a location tag in applicationHost.config that sets the delegation state of the configuration section by setting the **overrideMode** attribute to Allow or Deny. For example, if we use the unlock command shown previously to unlock the <serverRuntime> section, we will generate the following in applicationHost.config.

```
<location path="" overrideMode="Allow">
    <system.webServer>
        <serverRuntime />
    </system.webServer>
</location>
```

Likewise, you can lock or unlock configuration sections for a particular configuration path only, by specifying this path in the command. This can allow you, for example, to keep the configuration section locked for the entire server but allow a specific site to override its settings.

```
%windir%\system32\inetsrv\AppCmd Unlock Config "Default Web Site/"
/section:system.webServer/serverRuntime /commit:apphost
```

In this example, we unlock the <serverRuntime> section for the "Default Web Site" only and commit these changes to applicationHost.config (this is required). This produces a location tag in applicationHost.config that uses the *path* attribute to apply itself only to "Default Web Site/".

This enables you to quickly manage the configuration delegation on a section level. However, sometimes it is necessary to allow the delegation of the section but keep control over a specific setting inside that section. This can be accomplished using granular configuration locking, which we'll discuss in the section titled "Granular Configuration Locking" later in this chapter.

Additional Configuration for Remote Administration

For a user to manage a site or application remotely using the IIS Manager, it is necessary to assign specific permissions to the content. The service account for the Web Management Service (WMSvc) must have read and write permissions to web.config in order to successfully connect remotely. Please refer to Chapter 8 for these and other details.

Granular Configuration Locking

You have explored the configuration's ability to lock and unlock sections for delegation and used the location tag for creating settings for a site or directory that override the inherited defaults. Feature delegation controls whether or not the entire section can be used in a configuration file at a certain level. However, there are some cases in which the configuration section contains some configuration that should be delegated and some configuration that should be locked.

To support these scenarios, the configuration system allows you to exercise more fine-grained control over what specific configuration settings should be delegated through granular locking. Granular locking is achieved through the use of special locking directives supported by the configuration system.

To use granular configuration locking, you have to edit the configuration through some means other than the IIS Manager. At this time, the IIS Manager does not support configuring granular locking.

Note The semantics for granular locking are based on the configuration system for ASP.NET, so if you are familiar with that, you will be ahead of the game.

Granular configuration locking is accomplished by using one of the special attributes listed in Table 4-9.

Table 4-9 Granular Configuration Locking

Locking Directive	Used To
lockAttributes	Lock specific attributes to prevent them from being specified.
lockAllAttributesExcept	Lock all attributes on the element other than the specified attributes.
lockElements	Lock the specified elements to prevent them from being specified (and therefore lock all other attributes and child elements of the specified elements)
lockAllElementsExcept	Lock all elements on the current element except the specified elements.
lockItem	Lock the current collection element to prevent it from being removed.

lockAttributes, lockAllAttributesExcept The *lockAttributes* configuration directive can be specified on a configuration element in order to lock specific attributes on the element and prevent them from being specified at lower configuration levels. The *lockAttributes* directive specifies a comma-separated list of attribute names that are valid for the current element.

For example, in order to allow the <defaultDocument> section to be delegated but make sure that the feature itself cannot be disabled, we can set the *enabled* attribute to "true" and then lock it using the *lockAttributes* directive as follows:

```
<defaultDocument enabled="true" lockAttributes="enabled">
  <files>
    <add value="Default.htm" />
    <add value="Default.asp" />
    <add value="index.htm" />
    <add value="index.html" />
    <add value="iisstart.htm" />
    <add value="default.aspx" />
  </files>
</defaultDocument>
```

In this example, *lockAttributes* instructs IIS 7.0 to disallow any change to the *enabled* attribute. As a result, if the Web administrator attempts to turn off the default document feature (enabled="false") the error message shown in Figure 4-6 occurs.

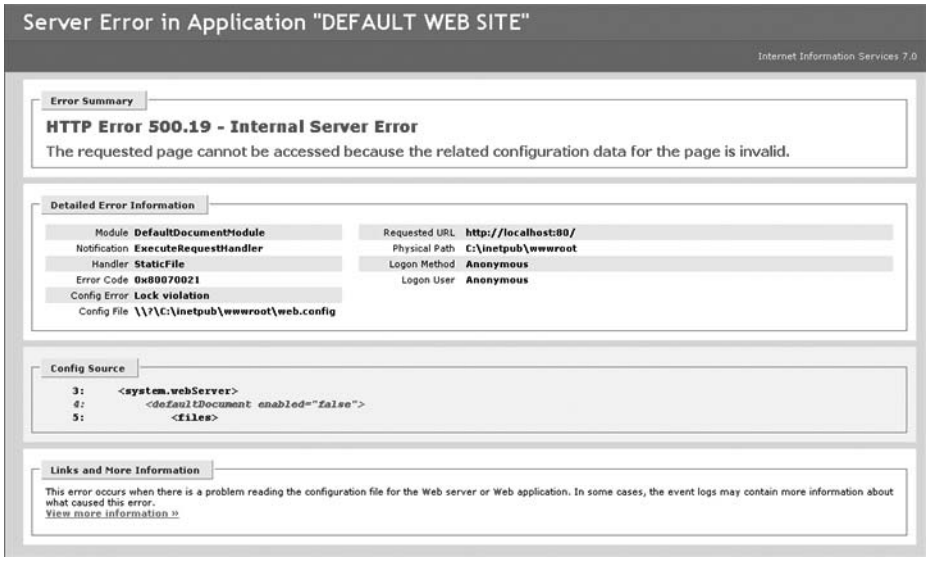


Figure 4-6 Error message due to configuration locking.

As you can see in Figure 4-6, the lock violation is called out and the offending line in web.config is clearly displayed. Removing this line, in this case, clears the error.

The *lockAllAttributesExcept* form of the attribute lock provides a convenient mechanism for cases in which you want to lock all attributes on the element except for one or two attributes that should be unlocked. In that case, you can use it instead of the *lockAttributes* element and specify the attributes that you want to keep unlocked.

lockElements, lockAllElementsExcept The *lockElements* locking directive allows you to lock a particular child element of the current element (as opposed to an attribute). This prevents this element from being specified at lower configuration levels. The *lockElements* directive specifies a comma-separated list of element names to lock.

For example, we can use the *lockElements* directive to prevent the <files> collection of the <defaultDocument> section from being specified, therefore effectively preventing lower configuration levels from changing the contents of the default document list.

```
<defaultDocument enabled="true" lockElements="files" >
  <files>
    <add value="Default.htm" />
    <add value="Default.asp" />
    <add value="index.htm" />
    <add value="index.html" />
    <add value="iisstart.htm" />
    <add value="default.aspx" />
  </files>
</defaultDocument>
```

This setup prevents a Web administrator from changing the files in the default document list, but it does permit turning the feature on and off (via the enabled attribute).

The *lockElements* directive can also be used to do collection locking. By locking the ability to use certain collection elements (such as <add>, <remove>, and <clear>) it is possible to prevent the collection from being changed or prevent elements from being removed while still allowing new elements to be added.

For example, if you lock the <add> element (or the corresponding element that acts as the <add> element for the collection), lower configuration levels will not be able to add new elements to the collection. Likewise, if you lock the <remove> and <clear> elements, lower levels will not be able to remove elements from the collection but will be able to add new ones.

The *lockAllElementsExcept* directive can be used with configuration elements that have multiple subelements, when you want to lock all of them but one. In practice, we don't expect that this will be widely used, but it is a possibility to keep in mind should you encounter a situation in which it is applicable.

lockItem The *lockItem* directive can be used to lock specific collection elements from being removed or modified, as opposed to preventing all elements in the collection from being removed by locking the <remove> element using *lockElements*. The *lockItem* directive is specified on each collection element that is to be locked and accepts Boolean values.

Returning to our example, we want to allow a Web site administrator to be able to add new entries to the list of default pages but not remove Default.aspx from the list. In applicationHost.config, you can lock in the Default.aspx page by finding the configuration section in applicationHost.config as follows.

```
<defaultDocument>
   <files>
       <add value="Default.htm" />
       <add value="Default.asp" />
       <add value="index.htm" />
       <add value="index.html" />
       <add value="iisstart.htm" />
       <add value="default.aspx" lockItem="true" />
   </files>
</defaultDocument>
```

This will prevent lower configuration levels from being able to explicitly remove the Default.aspx entry, as well as using <clear/> to remove all items from the collection. They will still be able to add new entries to the collection.

An important use of *lockItem* is implemented in applicationHost.config. If you examine the <modules> section, you'll notice that modules are added with *lockItem* set to "true." This means that if IIS 7.0 encounters a <clear> or <remove> in a web.config or location tag that references the locked module, you will get a locking error. These locks are enabled by default since delegation is enabled for modules in order to permit .NET applications to add modules, a feature that is quite common. However, by delegating the modules section, it is also possible to remove modules in web.config. This could allow a user to inadvertently create an insecure or nonfunctional configuration. To prevent this from occurring, while at the same time ensuring maximum compatibility with .NET, modules are declared with *lockItem* specified as true.

Sharing Configuration Between Servers

An entirely new feature in IIS 7.0 is the ability to have multiple Web servers share a single configuration file. This feature was designed with load-balanced Web farms in mind in order to eliminate the need to keep multiple server configuration in sync. Toward this end, shared configuration is an excellent feature that will be useful in many Web farm situations.

Note Shared configuration is not a complete Web farm solution in itself, because it does not eliminate the need to synchronize application content and local components like SSL certificates or .NET assemblies registered in the GAC.

Enabling Shared Configuration

You can enable shared configuration using the IIS Manager. You'll find the IIS Manager Shared Configuration icon in the features pane when the Server node is selected in the tree view. Look for it at the bottom in the Management section, as shown in Figure 4-7.

Note It is possible to enable shared configuration without using IIS Manager by modifying IIS configuration manually and performing all the necessary import steps. However, IIS Manager is recommended because it automates a lot of these steps and makes setting up shared configuration a lot easier than it otherwise would be.

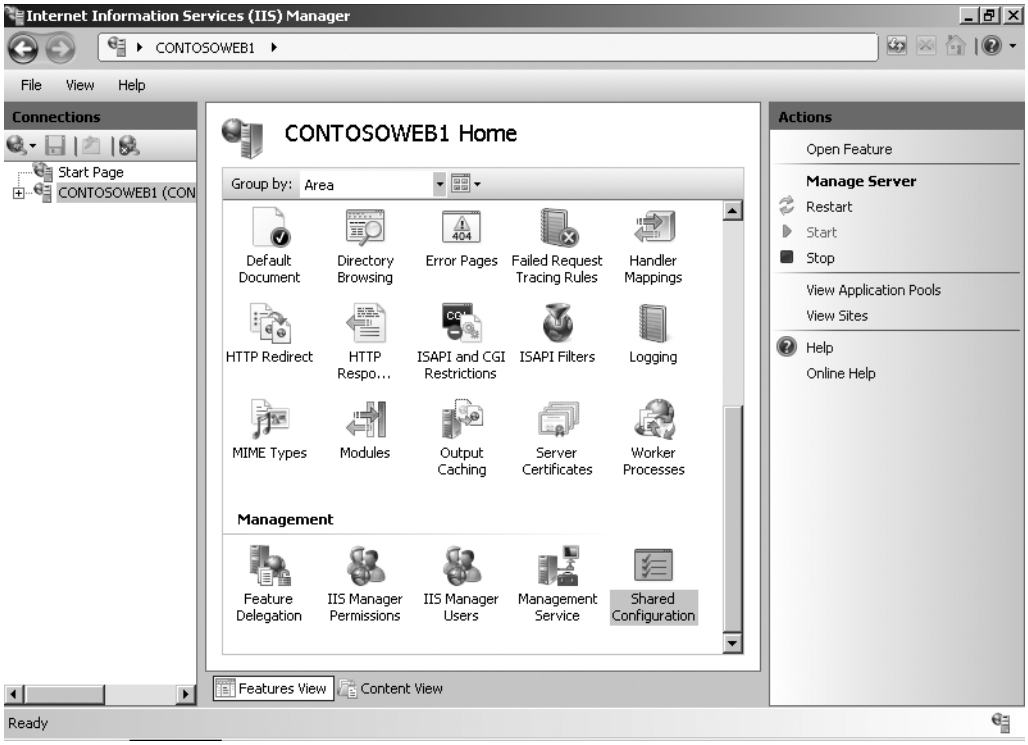


Figure 4-7 The Shared Configuration icon in IIS Manager.

How Shared Configuration Works

The basic notion behind shared configuration is to place the main configuration files for IIS 7.0 on a shared UNC path and have all the servers in the farm use the remote configuration store as if it were local. In addition, if you direct command line administration tools to modify settings on a server that uses the shared configuration, those instructions are redirected to the shared store. The net result is that if you have 10 servers sharing configuration, and you add an application pool, all 10 servers will have that pool immediately.

Setting up shared configuration involves three main actions: First, you have to create a location with the proper permissions and a user identity that will be used to access the content. Second, you must export the configuration files to a centralized location. Third, you have to set up the servers to use the shared configuration files instead of the local configuration files. At that point, they are all functionally identical.

Step 1: Preparing for Shared Configuration The IIS Manager has to write to the remote configuration as a user of some kind, so it must be provided with the credentials of a local or domain user that has the correct permissions. So the first task is to create a user that has the correct permissions and then assign NTFS permissions for that user to the shared location.

1. Create a user that you will use to provide read and write access to the shared configuration files. This can be a local user that has the same credentials on each server, or a domain user presuming all the servers are joined to a domain.

   ```
   net user ConfigAccess HighSecurePasswordhere /add
   ```

2. Create a folder that will contain the shared configuration files. This can be on one of the Web servers or the file server. The only real requirement is that it be accessible via a standard UNC share from all the servers.

3. Configure the folder for sharing with the appropriate share permissions. We'll use the SharedConfig folder in this example.

   ```
   Net share sharedconfig$=%SystemDrive%\sharedconfig /grant:ConfigUser,Read
   /grant:Administrators,Full /grant:System,Full
   ```

4. Carefully inspect the configuration of the server you plan to use as the source for the shared configuration. The IIS 7.0 configuration you export will be shared by all the other servers, so take some time to make sure it is correct. You can, of course, change it after you've enabled shared configuration, but the changes will affect multiple servers at that time.

5. Back up the existing configuration files with the following commands from an administrative command prompt.

   ```
   windir%\system32\inetsrv appcmd add backup SharedConfigBackup
   ```

Step 2: Export the Configuration Files

1. In the IIS Manager, click the server node and then double-click the Shared Configuration icon.

2. In the Actions pane, click Export Configuration to open the Export Configuration dialog box, as shown in Figure 4-8.

3. Under Configuration Location in the Physical Path text box, enter the UNC path to shared configuration.

Note You can export the configuration files to a local, nonshared path if you prefer and then manually copy the files to the shared location.

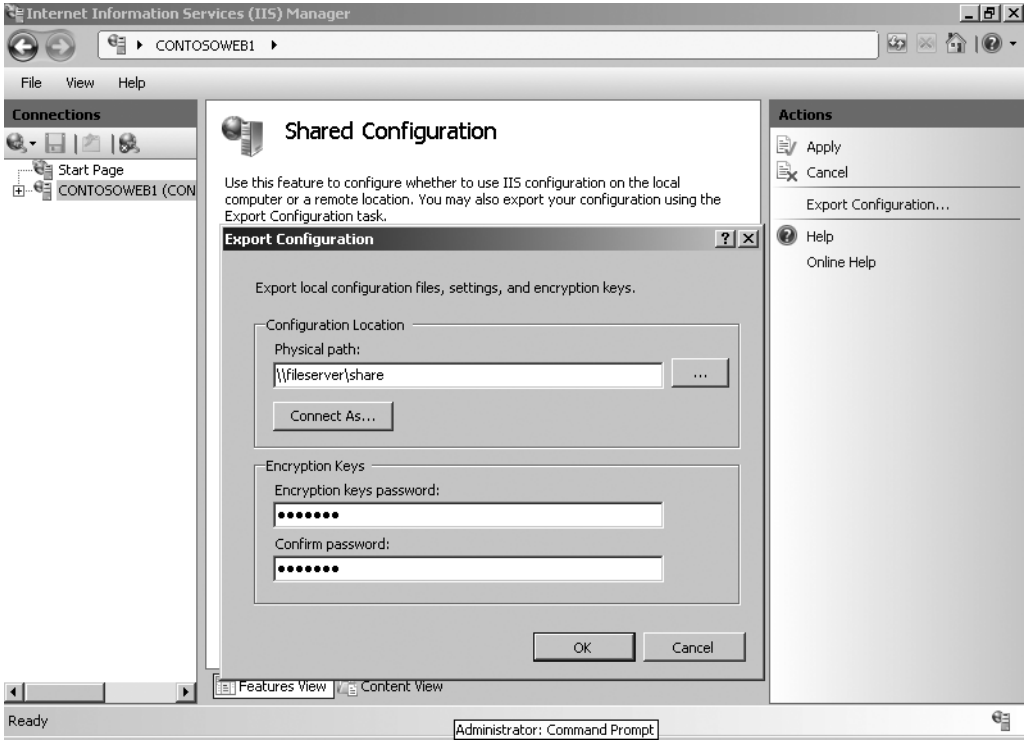


Figure 4-8 The Export Configuration dialog box.

4. Click Connect As and enter the credentials that have write access to the share. You could also enter administrative credentials here. These credentials are just used to write the configuration file in this export step and are not used for regular access to the shared configuration settings.

> **Caution** Do not use the ConfigAccess credentials you created for accessing the configuration from the Web server. These credentials should not have write access to the share.

5. Under Encryption Keys, enter a password that will be required to protect the exported encryption keys when transported off the server. You will need to provide this password on any server that will use the shared configuration files so that it can import the exported encryption keys. Note that the password must be at least eight characters, have a symbol, mixed case, and a number before it will be accepted. At this time, creating the encryption key cannot be automated.
6. Press OK. You will see a message that says the export was successful.

At this point you have not yet enabled shared configuration, just created a set of files that could be used for shared configuration. Before you proceed, you might want to see what was (and was not) copied. See the sidebar titled "Inspecting the Exported Configuration Files" for more information.

Inspecting the Exported Configuration Files

Open the location where you exported the files and examine the contents. You'll find a copy of applicationHost.config, administration.config, and an encrypted file named ConfigEncryptedKey.key. The .key file is used to decrypt any secrets stored in the .config files. For this to work, all the servers in the farm have to know a shared secret, and that's the reason for a strong key to be entered when you export these configuration settings. By default, there are no secrets in the config files, because the anonymous user is now a built-in account and no longer requires a password since it cannot be used to log on to the server. In addition, the IWAM account found on IIS 4.0, IIS 5.0, and IIS 6.0 is deprecated. However, many companies use unique identities as principals for application pools and the IIS anonymous user in order to increase security and provide more granular details in audit logs.

Passwords created in the IIS manager associated with UNC paths, applications pool principals, and custom anonymous users are encrypted and stored in the configuration files. These encrypted items cannot be deciphered by default on other IIS 7.0 servers. Shared configuration, however, makes this possible by allowing you to export the encryption keys from the server whose configuration is being exported and reimport them to all other machines using the shared configuration.

You should note that you will not see any web.config, custom modules, Web site content, certificates, or other files that are related to the server configuration. Centralized configuration enables sharing of applicationHost.config and administration.config files only. All other items needed to keep the servers functionally identical need to be managed by processes you institute outside of IIS 7.0.

Note that you can expect to see tools or updates to this feature from the IIS team after Windows Server 2008 is released that will help with replication and synchronization tasks.

Step 3: Enable Shared Configuration You're now ready to enable shared configuration. Typically, you'll start with the server used for the shared configuration export. Exporting the configuration does not automatically cause the server to start using the exported settings. In fact, if you make any changes at this point to the IIS 7.0 configuration, you will see them on the local server, but unless you re-export the configuration, the shared configuration will not have the most recent changes.

The procedure is simple and is the same for each server:

1. In the Shared Configuration feature, select the Enable Shared Configuration check box.

2. Enter the Physical Path, User Name, and Password you used to create the share and user for the share. In our example, this would be:

   Path: **\\Contoso\SharedConfig**
   User: **ConfigAccess**
   Password: **HighSecurePasswordHere**

3. At the prompt, enter the password you used to export the settings. You will see the message shown in Figure 4-9.

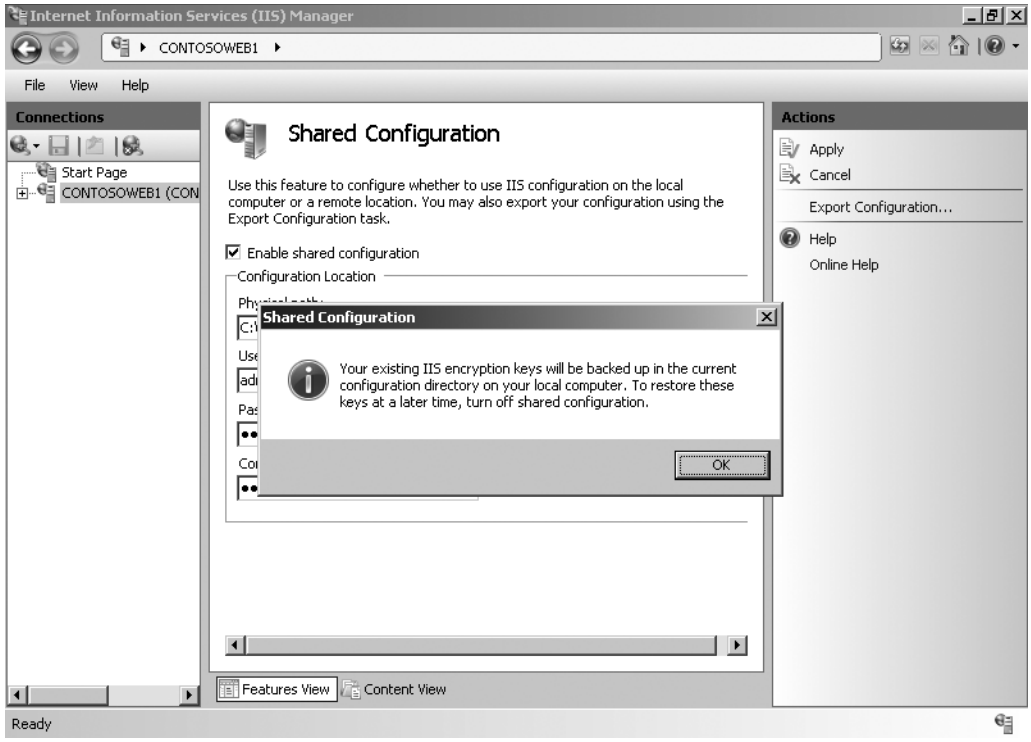


Figure 4-9 Backing up encryption keys.

   This message informs you that if you decided to revert back to your local settings, the IIS Manager will fix your encryption keys so they will work on your local configuration files. Otherwise, any passwords you entered in the configuration system for UNC paths, custom anonymous users, or application pool identities could not be deciphered by IIS.

4. Click OK to close the message box. You will see another message that says you need to close and reopen the IIS Manager and reset (stop and start) the Web Management Service for changes to take effect. When you close and reopen the IIS Manager, you will load the redirected configuration files instead of the local files. Restarting the Management Service will cause remote administration requests to be redirected.

You will need to repeat this procedure on each server.

Shared Configuration Considerations

Shared configuration will help to reduce the administrative burden of configuration replication between servers in a Web farm. It is not, however, a Web farm management tool. You will still need to manage replication of any content or configuration item that is local to a server in the farm. This typically involves such items as content replication, directory structure maintenance, SSL certificates, recycling of services, operating system updates, registering COM objects, placing new content in the .NET global assembly cache, network configuration, and other settings that are stored locally.

Consider the scenario in which you want to change the type of application pool from Classic to Integrated. This is one of the few settings that will cause an application pool to recycle. Making changes that affect the application pool environment, such as the application pool type or pool identity, will cause all of your application pools to recycle across all the shared servers, potentially resulting in your Web application becoming unavailable for a short period of time. As a result of this and other scenarios such as content updates, you will want to devise a method for rolling in updates so that you can more precisely control the settings.

For example, if you need to make a configuration change that would result in a recycle, you should export the settings from a second server to a new shared location. This server will be the only one using that location while the other servers in the farm continue to deliver requests. You then make the updates you want to this server and test the results. When you are satisfied, you move the other shared servers to the new shared location in series. If at any time you don't like what's going on, you can roll back to the prior configuration. If things proceed well, you will continue moving each server over until all the servers are using the new configuration.

Summary

The IIS 7.0 configuration system is the foundation for many key deployment and management capabilities of the server. For the first time, it enables scenarios including delegated management of configuration, true xcopy deployment of IIS applications, and sharing configuration between multiple servers.

In this chapter, you reviewed the basics of editing Web server configuration, and performing key configuration management tasks such as backing up configuration and setting up shared configuration for multiple servers on a Web farm.

The configuration system is the core of the IIS 7.0 Administration stack, which offers a full set of options for managing the server. To learn how to manage IIS using GUI, see Chapter 6. You can also learn about managing IIS 7.0 configuration from the command line in Chapter 7.

In the spirit of IIS 7.0 end-to-end extensibility, the IIS 7.0 configuration system is also completely extensible, allowing third-party Web server modules to store their configuration in the IIS 7.0 configuration system. This extensibility allows developers to make use of the same

configuration capabilities and management tools used by IIS 7.0. For example, an administrator could configure the custom feature using Appcmd or a developer could use .NET to manage the feature state and configuration.

For more information about protecting configuration on your server, including using configuration encryption and properly taking advantage of configuration isolation, see Chapter 14.

Additional Resources

These resources contain additional information and tools related to this chapter:

- The IIS 7.0 Web Reference can be found at *http://msdn2.microsoft.com/en-us/library/ms691259.aspx*.
- You can search for configuration information on the IIS Web site at *http://www.iis.net* and in the IIS 7.0 online help files.
- You'll find information about how IIS 6.0 metabase properties map to IIS 7.0 configuration schema at *http://msdn2.microsoft.com/en-us/library/aa347565.aspx*.
- For more information about configHistory, see the article "Using IIS7 Configuration History" at *http://www.iis.net/articles/view.aspx/IIS7/Managing-IIS7/Configuring-the-IIS7-Runtime/Understanding-AppHost-Service/Using-IIS7-Configuration-History?Page=1*.