Part I: Introduction and Deployment

1

Background on IIS and New Features in IIS 7.0

Microsoft's Internet Information Services (IIS) has been around for more than a decade, from its first incarnation in Windows NT 3.51 to the current release of IIS 7.0 on the Windows Server 2008 and Vista platforms. It has evolved from providing basic service as an HTTP server, as well as additional Internet services such as Gopher and WAIS, to a fully configurable application services platform integrated with the operating system.

IIS 7.0 is a dramatic change in the way IIS is configured and managed. Modularity, granularity, and interoperability are the guiding factors across the entire product, from setup to security, management to automation. Integrated heavily into the operating system, IIS 7.0 benefits from the improvements in the Windows Server 2008 operating system but IIS has been re-engineered to meet the demands of a true application platform.

This chapter will provide you with an overview of the changes in IIS 7.0 as well as a sampling of some of the new technologies. If you are familiar with IIS 6.0, you will want to skim through this chapter for changes before digging into future chapters for specifics. If you are new to IIS, this chapter will provide an introduction to the features in IIS 7.0 and provide you with a basis for understanding future chapters. And if you're the kind of reader who just wants to skip to the part that applies to your immediate needs, this chapter can help you figure out in what area those needs will lie.

IIS Versions 1.0 to 4.0

IIS was released with Service Pack 3 for Windows NT 3.51, as a set of services providing HTTP, Gopher, and WAIS functionality. Although the functions were there, most users chose alternates from third-party vendors such as O'Reilly's Website or Netscape's server. Although these services had been available for years with the various flavors of UNIX operating systems, native Internet services for Windows were mostly an afterthought, with little integration with the Windows operating system.

With the advent of Windows NT 4.0, IIS also matured in version 2.0. The most notable improvement in IIS version 2.0 was closer integration with the Windows NT operating system, taking advantage of Windows security accounts and providing integrated administration through a management console similar to many other Windows services. IIS 2.0 introduced support for HTTP Host headers, which allowed multiple sites to run on a single IP address, and aligned Microsoft's IIS development with NCSA standards, providing for NCSA common log formats and NCSA-style map files. IIS 2.0 also introduced a web browser interface for management, and content indexing through Microsoft's Index Server.

IIS version 3.0 was introduced with Windows NT Service Pack 3 and introduced the world to ASP (Active Server Pages) and Microsoft's concept of an *application server*. A precursor to the ASP.NET environment, ASP (now referred to as *classic ASP*) is a server-side scripting environment for the creation of dynamic web pages. Using VBScript, JScript or any other active scripting engine, programmers finally had a viable competitor to CGI and scripting technologies available on non-Microsoft platforms, such as Perl.

IIS 4.0, available in the NT Option Pack, introduced ASP 2.0, an object-based version of ASP that included six built-in objects to provide standardized functionality in ASP pages. IIS 4.0 was the last version of IIS that could be downloaded and installed outside of the operating system.

IIS 5.0 and 5.1

With the release of Windows 2000, IIS became integrated with the operating system. Version numbers reflected the operating system, and there were no upgrades to IIS available without upgrading the operating system. IIS 5.0 shipped with Windows 2000 Server versions and Windows 2000 Professional, and IIS version 5.1 shipped with Windows XP Professional, but not Windows XP Home Edition. For all essential functions, IIS 5.0 and IIS 5.1 are identical, differing only slightly as needed by the changes to the operating system.

With Windows 2000 and IIS 5.0, IIS became a service of the operating system, meant to be the base for other applications, especially for ASP applications. The IIS 5.0 architecture served static content, ISAPI functions, or ASP scripts, with ASP script processing handed off to a script engine based on the file extension. Using file extensions to determine the program that handles the file has always been a common part of Windows functionality, and in the case of ASP processing, the speed of serving pages was increased by the automatic handoff of ASP scripts directly to the ASP engine, bypassing the static content handler. This architecture has endured in IIS to the current version.

IIS 6.0

IIS 6.0 shipped with Windows Server 2003 editions and Windows XP Professional 64bit edition, which was built on the Windows Server 2003 Service Pack 1 code base. IIS 6.0 was identical among operating system versions, but there were restrictions or expansions depending on the version of Server 2003 under which IIS was running. For example, Server 2003 Web Edition would only run IIS and a few ancillary services; it could not be used to run Microsoft SQL Server. On the other end of the spectrum, only the Enterprise and Data Center versions of Server 2003 included clustering technology.

Operating system changes also expanded the capabilities of IIS as an application server. Native XML Web Services appeared in Server 2003. Process-independent session states made web farms easier to configure and manage, allowing session states to be stored outside the application for redundancy and failover. Web farms also became easier with Server 2003's improved Network Load-Balancing features, such as the NLB Manager, which provided a single management point for NLB functions.

Secure by Default

Windows Server 2003 and IIS 6.0 shipped in a secure state, with IIS no longer installed by default. Even when IIS was installed, the default installation would serve only static HTML pages; all dynamic content was locked down. Managed through Web Service Extensions, applications such as ASP and ASP.NET had to be specifically enabled, minimizing default security holes with unknown services open to the world.

IIS 6.0 also ran user code under a low privilege account, Network Service, which had few privileges on the server outside of the IIS processes and the web-site hierarchy. Designed to reduce the damage exposure from rogue code, access to virtual directories and other resources had to be specifically enabled by the administrator for the Network Service account.

IIS 6.0 also allowed delegation for the authentication process; thus administrators and programmers could further restrict account access. Passport authentication was also included with IIS 6.0, although in real-world use, it never found widespread favor among administrators. Kerberos authentication, on the other hand, allowed secure communication within an Active Directory domain and solved many remote resource permission issues.

IIS 6.0 also would serve only specific file requests, by default not allowing execution of command-line code or even the transfer of executable files. Unless the administrator assigned a specific MIME type to be served, IIS would return a 404 error to the request, reporting the file not found. Earlier versions of IIS included a wildcard mapping and would serve any file type.

Request Processing

IIS 6.0 changed the way IIS processed requests, eliminating what had been a major performance hurdle in scaling prior IIS versions to serve multiple sites. IIS 6.0 used the Http.sys listener to receive requests, and then handed them off to worker processes to be addressed. These worker processes were isolated to application pools, and the administrator could assign application pools to specific sites and applications. This meant that many more requests could be handled simultaneously, and it also provided for an isolated architecture in cases of error. If a worker process failed, the effects would not be seen outside the

application pool, providing stability across the server's sites. In addition, worker processes could be assigned a processor affinity, allowing multiprocessor systems to split the workload.

Additional Features

As did its predecessors, IIS 6.0 included additional features and functionality. Some internal features, such as HTTP compression and kernel mode caching, increased performance of the web server and applications served from it. Other features affected configuration, such as the move to an XML metabase, or stability, such as being able to configure individual application pools and isolate potential application failures. Still others added or expanded utility and ancillary functions, such as the improved FTP services or the addition of POP services to the existing SMTP service.

HTTP Compression

IIS 6.0 extended HTTP compression over the minimal compression allowed in previous versions. The HTTP 1.1 specification includes HTTP compression, and IIS 6.0 supported it for both static and dynamic content. Available in IIS 5.0 as an ISAPI add-on for the entire site, HTTP compression in IIS 6.0 became an integrated feature granularly controllable down to the specific file.

Kernel Mode and Persistent Caching

IIS 6.0 added a kernel mode cache to increase performance for dynamic content. In previous versions, static content was cached and served from cache whenever possible, but in IIS 6.0 caching was expanded to include dynamic content. In addition, whereas ASP templates were formerly cached in memory when allocated, IIS 6.0 added a persistent cache so that de-allocated templates were written to disk for future reallocation, as needed. Caching heuristics were also used to determine what to cache and when.

XML Metabase

The metabase, where IIS configuration settings are stored, was a binary file prior to IIS 6.0. Changed to an XML file, the metabase in IIS 6.0 could be edited while the site was active, and, although many functions wouldn't change until IIS was recycled, changes were in plain text. The XML metabase in IIS 6.0 was unstructured and not well documented though, and several functions still resided in registry settings, all of which gets changed in IIS 7.0.

Application Pools

IIS 6.0 changed the way applications behaved in memory, isolating applications into memory pools. Administrators could configure separate memory pools for separate applications, thus preventing a faulty application from crashing other applications outside its memory pool. This is particularly important in any shared web server environment, especially with ASP.NET applications.

FTP Service

The FTP service grew up in IIS 6.0, providing for greater security and separation of accounts through a new isolation mode using either Active Directory or local Windows accounts. Using Windows accounts or Active Directory accounts, users could be restricted to their own available FTP locations without resorting to naming the home directories the same as the FTP accounts. In addition, users were prevented from traversing above their home directories and seeing what other accounts may exist on the server. Even without NTFS permissions to the content, security in FTP before IIS 6.0 was still compromised because a user could discover other valid user accounts on the system.

The FTP service that ships with Windows Server 2008 is exactly the same as shipped in Server 2003. However, the Microsoft IIS development team is also shipping a new FTP server that includes many of the enhancements requested over the years. This server ships as a free download from `www.iis.net`*, as will many supported and unsupported tools. For more about configuring FTP, see Chapter 10, "Configuring Other Services."*

SMTP and POP Services

The SMTP service in Windows Server 2003 didn't change much from previous versions, allowing for greater flexibility and security but not altering the core SMTP functions. Most administrators would not use the SMTP service in IIS for anything other than outbound mail, instead relying on third-party servers or Microsoft's Exchange Server for receiving and distributing mail. But the addition of a POP3 service in Server 2003 allowed a rudimentary mail server configuration, useful for testing or small mail domains. Although SMTP can be used to transfer mail, most mail clients such as Microsoft Outlook rely on the POP3 or IMAP protocols to retrieve mail, which was unavailable without additional products until Windows Server 2003 and IIS 6.0.

IIS 7.0 Versions

Although there is really only a single version of IIS 7.0, the availability and capabilities vary with the choice of operating system. Because IIS 7.0 is tied to the operating system, as were all versions of IIS since IIS 4.0, it is not available on operating systems prior to Vista or Windows Server 2008. As in Windows XP, the workstation operating systems have limited IIS functionality or no functionality at all. Unlike in Windows XP, Vista versions have no concurrent HTTP connection limitations but instead use concurrent request processing limitations. In XP, reaching a maximum concurrent HTTP connection limit would result in IIS returning a 403.9 result code (too many users), while a request limitation merely queues requests in the order received. The end result is slower response, but no errors.

Some Windows Server 2008 versions also have limitations that affect IIS 7.0. Although IIS 7.0 is included with no limitations in all server versions, the server version itself may have limitations in use. For example, if you install Windows Server 2008 Core Edition, IIS 7.0 can be installed, but there is no GUI configuration application. This version of Windows does not have the .NET Framework available; thus, no managed code can be run on the server.

Windows Server 2008 Web Edition has no functional limits to IIS, but only supports three role services: Web Server, Windows Media Server and Sharepoint Services – it cannot be used to host a Domain Controller, or other types of roles. The following table shows which versions of Windows Vista and Windows Server 2008 have IIS 7.0 and what the limitations are.

Windows Version	IIS 7.0 Included	Limitations
Vista Starter Edition	No	No IIS functions available.
Vista Home Basic Edition	No	Contains some IIS functions, such as HTTP processing, but cannot be used as a web server.

Continued

Windows Version	IIS 7.0 Included	Limitations
Vista Home Premium Edition	Yes	Limited to three concurrent requests; no FTP server.
Vista Business Edition	Yes	Limited to 10 concurrent requests.
Vista Enterprise Edition	Yes	Limited to 10 concurrent requests.
Vista Ultimate Edition	Yes	Limited to 10 concurrent requests.
Vista Home Basic N Edition[1]	No	Contains some IIS functions, such as HTTP processing, but cannot be used as a web server.
Vista Business N Edition	Yes	Limited to 10 concurrent requests.
Server 2008 Core	Yes	No GUI management interface and no .NET Framework.
Server 2008 Web Edition	Yes	Supports Web Server, SharePoint, and Windows Media Server roles.
Server 2008 Standard Edition	Yes	None

1 The N editions of Windows Vista are for release in the European Union and do not include an embedded Windows Media Player.

IIS 7.0 Features

IIS 7.0 is a ground-up rewrite of IIS 6.0, designed as an integrated web application platform. Integration with the ASP.NET framework combined with fully exposed APIs for complete extensibility of the platform and management interfaces make IIS 7.0 a programmer's dream. Security that includes delegation of configuration and a complete diagnostic suite with request tracing and advanced logging satisfies the administrator's desires.

While the most substantial change in IIS 7.0 may be the integration of ASP.NET into the request pipeline, the extensibility of IIS 7.0, configuration delegation and the use of XML configuration files, request tracing and diagnostics, and the new administration tools are all welcome changes from previous versions of IIS.

Unlike previous versions of IIS, the modular design of IIS 7.0 allows for easy implementation of custom modules and additional functionality. This increased functionality can come from in-house develop-

ment, third-party sources, or even Microsoft. Since these modules and additional programs can be plugged into IIS at any time, without changing core operating system functions, the Microsoft IIS development team can ship additional supported and unsupported modules outside of Microsoft's standard service pack process. The bottom line is that you get what you need faster. Microsoft's web site at `www.iis.net` is the source for these additional downloads.

Integrated Request Pipeline

One of the most radical changes in IIS 7.0 is its close integration with ASP.NET and the ASP.NET processes. There is a unified event pipeline in IIS 7.0. This pipeline merges the existing two separate IIS and ASP.NET pipelines that existed in IIS 6.0 and earlier. ASP.NET HTTP modules that previously only listened for events within the ASP.NET pipeline can now be used for any request. For backwards compatibility, a Classic pipeline mode exists, which emulates the separate IIS and ASP.NET pipeline model from IIS 6.0

In the IIS 6.0 model, an HTTP request first would be checked for the required authentication and then passed on through the pipeline. The request would be evaluated for any ISAPI filters that had been installed, such as processing by URLScan; the cache was checked to see if the request already existed; and if the request could not be served from the cache, the file extension was evaluated to determine the appropriate handler for the request. For example, if the extension was .SHTM, the server-side includes process was invoked, additional code was inserted in the page being served, and then that page was processed as an HTML request and sent along the pipeline, the response was logged, and eventually the requested page was returned to the browser.

If that requested file was an ASP.NET file with an .ASPX extension, then the process grew even more complicated, as shown in Figure 1-1. The request was shunted to the ASPNET_ISAPI.DLL and began a process through the ASP.NET pipeline. The request was again evaluated for authentication and processed for ASP.NET caching; the appropriate ASP.NET handler was determined; and the request was processed, cached, logged, and handed back to the Http.sys pipeline for completion and serving to the requesting browser.

This process occurred because the architecture consisted of a single, monolithic DLL with all the components loaded. ISAPI extensions were also single DLLs, as were any CGI processes, and each had to be loaded in memory and each request processed through the same DLL. IIS 6.0 allowed application pools and separate implementations of the Http.sys process, which increased overall performance, but there was no way to tune the core server operations themselves.

In IIS 7.0, there is a single, unified pipeline, as shown in Figure 1-2. Forms authentication and role management from ASP.NET are part of the authentication and authorization process; thus a request is authenticated a single time. In IIS 7.0, all requests can be processed through the ASP.NET Forms authentication module, not just those requests for files ending in an .ASPX extension. A request for `www.domain1.com/images/myimage.gif` will pass through the ASP.NET Forms authentication process, and if an authentication constraint in the `web.config` prevents serving that file or folder, unauthorized users will be unable to view or download the image. Requests now pass through the pipeline and exit, served to the requesting browser, without having to branch into ISAPI processes like ASP.NET. Although the ISAPI handlers exist for compatibility with existing code, the request doesn't need to be processed through ISAPI, and you don't even need to load the handlers if you don't need the compatibility for legacy code. Programmers may find themselves moving away from ISAPI now that the more familiar managed code of ASP.NET is available to meet the same needs.

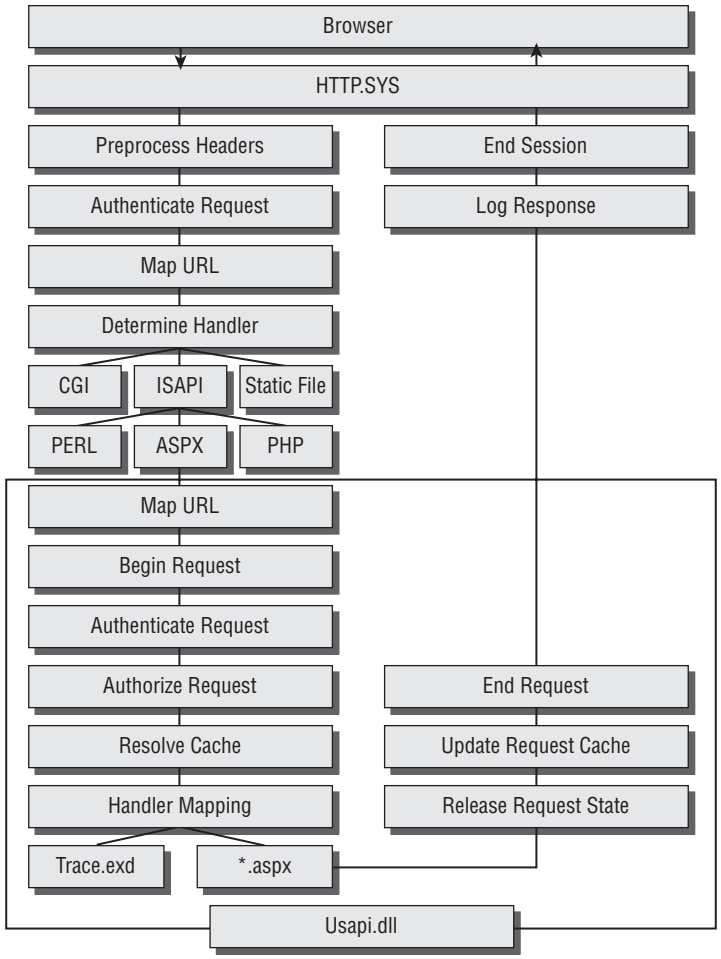

Figure 1-1

Within the IIS 7.0 pipeline, each process is handled by an individual component. These components can be specifically loaded for those sites that need them, and left out of the pipeline for sites and applications that do not. The components are configurable at the application, site, and server levels, and the ability to configure components can be delegated at any of those levels. In addition, custom components can be inserted into the pipeline, and even the order of components in the pipeline can be rearranged — for example, triggering a log trace at the start of the request and writing that log trace to a file as the request finishes processing. The order of execution is simply the order of the components in the configuration file. Components and their functions, as well as programming your own components, are covered in later chapters.

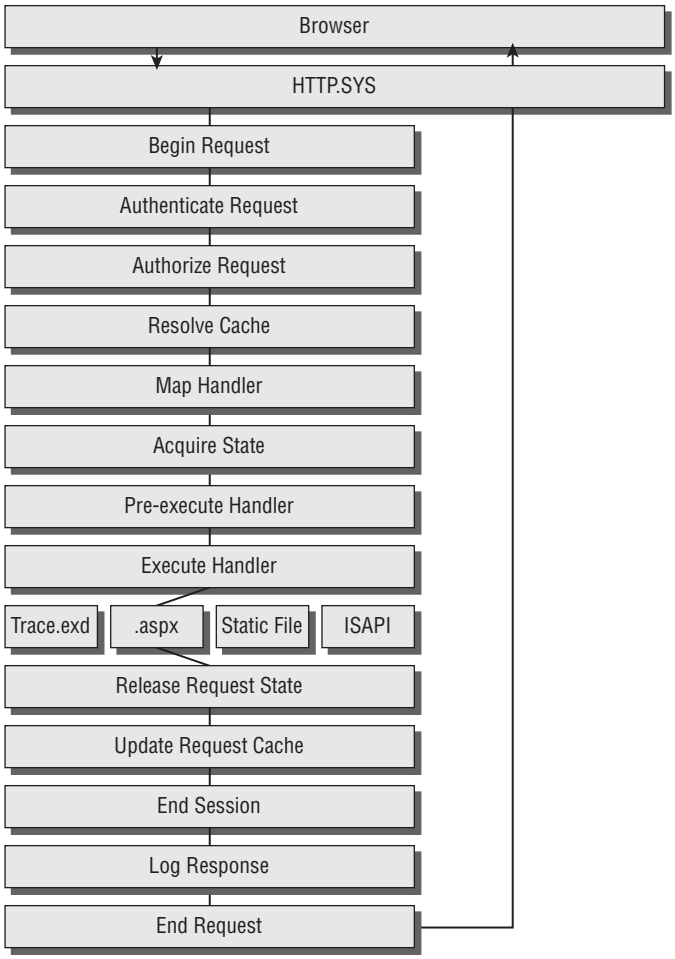

Figure 1-2

Configurability

Another and more visible change is the integration of IIS configuration into the same process used for configuring ASP.NET applications. Gone are the IIS registry settings, and the metabase that has been the repository of IIS configurations in previous versions has been replaced by XML-based configuration files that store both IIS and ASP.NET settings. This integration not only erases the line between ASP.NET applications and the application server on which they run, but it also allows for better configurability and easier deployment of both sites and applications. It also makes deployment across multiple systems in web farms more straightforward and allows for extensibility of the configurations. IIS 7.0 introduces the concept of shared configuration, wherein multiple web servers can point to the same physical file for configuration, making deploying configuration changes to web farms nearly instantaneous.

IIS 7.0 now stores settings in a new applicationHost.config file. Additionally, IIS 7.0 configuration options for individual websites or web applications can be stored in web.config files alongside ASP.NET settings, in a new system.webServer section.

Using applicationHost.config

The `applicationHost.config` file, new in IIS 7.0, stores IIS configurations for both the web server and the process model. Global configurations are now stored in the `%windir%\system32\inetsrv` folder in `applicationHost.config`. This file has two primary sections:

- `system.applicationHost` — Contains the configurations for the site, application, virtual directory, and application pools.
- `system.webServer` — Contains the rest of the settings and the global defaults.

Configurations by URL location can also be in `applicationHost.config` or in the `web.config` files for those locations. This allows administrators to set location defaults on the server, while developers can be allowed to override those settings as needed. These settings are inherited by the `web.config` files at both the root and application levels. This becomes important in the delegation of settings, since the IIS administrator can allow developers control over settings in a very granular manner at the application level, while retaining control for the site level.

The `applicationHost.config` file can be used to change the characteristics of an IIS server or site after IIS has been installed. For example, if you choose to use Windows authentication in an IIS site, you can use the IIS Manager to add Windows authentication, similar to the manner required with previous versions of IIS. Or you can use the following code within the `applicationHost.config` file to accomplish the same task for the site named *MyWebSite*:

```
<location path="MyWebSite">
   <system.webServer>
      <security>
         <authentication>
            <windowsAuthentication/>
         </authentication>
      </security>
   </system.webServer>
</location>
```

Similarly, adding ASP to a site is as simple as

```
<system.webServer>
   <asp/>
</system.webServer>
```

Configuring application pools is as easy as

```
<system.applicationHost>
   <applicationPools>
      <applicationPoolDefaults>
         <processModel
            userName="Site1AppPoolUser"
            password="Passw0rd"
         />
      </applicationPoolDefaults>
      <add name="Site1AppPool"/>
   </applicationPools>
</system.applicationHost>
```

There are two great benefits to this new configuration style in IIS 7.0. The first is that by not using the registry for configuration, deploying a site and applications can be done by using XCopy to transfer both content and configuration settings. This makes deployment across web farms far easier than trying to export/import metabase settings. It also speeds deployment to remote servers and provides for simplified customer installations of custom web-site applications that include specific web-site configurations.

The second benefit to this process is that developers can be allowed to modify the configuration files for their applications, determining the IIS configuration requirements necessary. This modification can be delegated so that required settings cannot be changed, and the settings are hierarchical, thus server settings cascade to the site and on to the application level, pending modifications allowed at lower levels. Developers accustomed to using configuration files for their applications need not learn IIS administration, and administrators can allow developers the flexibility they need while still maintaining overall control.

Extensible Configuration Schemas

IIS 7.0 configurations can be extended quite easily with the new configuration model. Suppose you want to create a new module for IIS. You would need to point to the module's DLL in the `<globalModules>` section of the `applicationHost.config` file and declare the module in either the `applicationHost.config` or the appropriate `web.config` file. Extending the configuration schema for your new module is as simple as creating the schema file in the `inetsrv\config\schema` folder on the system. Getting IIS to recognize and use the schema is done by adding a section for the module under the `<configSections>` section of `applicationHost.config`.

For example, you might add the following to the `<globalModules>` section:

```
<globalModules>
  <add name="MyNewModule" image="c:\modules\MyNewModule.dll" />
  ....
</global Modules>
```

The following would need to be added to the `<modules>` section of the `applicationHost.config` file or to the `web.config` file for the individual site in which the module would be used:

```
<modules>
  <add name="MyNewModule" />
  ....
</modules>
```

Then you would need to create a new schema file, `MyNewModule.xml`, in the `inetsrv\config\schema` folder for your new module:

```
<configSchema>
   <sectionSchema name="MyNewModule">
      <attribute name="enabled" type="bool" defaultValue="false" />
      <attribute name="message" type="string" defaultValue="Hello World!" />
      </sectionSchema>
</configSchema>
```

Finally, you need to register the section on the system in `applicationHost.config`, as follows:

```
<configSections>
```

```
    <section name="MyNewModule" />
    ....
</configSections>
```

With these simple changes to the configuration files, you've added the custom module MyNewModule to IIS, with its own custom schema.

Componentization

Extensibility doesn't only apply to configurations. Because of the changes to the request processing pipeline, the core server itself is extensible, using both native and managed code. This extensibility comes from the componentization of the core IIS functions. Instead of having to work with ISAPI filters to modify the request process, you can now inject your own components directly into the processing pipeline. These components can be your own code, third-party utilities and components, and existing Microsoft core components. This means that if you don't like Microsoft's Windows authentication process, you can not only choose to use forms authentication on all files, but also you can choose to bypass all built-in authentication and roll your own. This also means that if you don't need to process classic ASP files, you can simply not load that component. Unlike in previous versions, where components were loaded into memory in a single DLL, you can reduce the memory footprint of IIS 7.0 by not loading what you don't need.

Security

Componentization also increases the already strong security that existed in IIS 6.0. A perennial complaint against Microsoft had always been that IIS installed by default and that all services were active by default. IIS 6.0 and Server 2003 reversed that course — almost nothing was installed by default, and even when you did install it, the majority of components were disabled by default. To enable ASP.NET, you had to choose to allow ASP.NET as a web service extension. Classic ASP had to be enabled separately, as did third-party CGI application processors such as Perl or PHP.

With the exception of third-party software, though, IIS 6.0 still loaded all the services into memory — it just loaded them as disabled. For example, if you didn't want to use Windows authentication, as would be the case if you were using your own authentication scheme, you could choose not to enable it, but the code still resided in memory. Similarly, default IIS 6.0 installations were locked down to processing static HTML files, a good choice from a security standpoint. But what if you were never going to use static HTML files in your application or site? In IIS 7.0, you have the option of never loading the code in the first place.

This book devotes three chapters to security-related issues. In Chapter 13, securing your server is discussed. In Chapter 14, authentication and authorization are covered. Finally, in Chapter 15, SSL, and TLS are discussed. General Windows and network security precautions are not a major part of this book, but remember that IIS doesn't operate in a vacuum. Security risks need to be mitigated in all areas of your network infrastructure as well as all applications on your servers. A SQL Server breach won't technically be a compromise of your IIS security, but if the server is compromised, it really doesn't matter that it wasn't an IIS configuration, does it?

Minimal Installation

IIS 7.0 continues the tradition of its predecessor with minimal installation the default. IIS is not installed with the default operating system install, and a basic install only selects those options needed for serving static HTML files. The installation GUI for IIS 6.0 allowed a choice of eight different options, including installing FTP, whereas IIS 7.0's setup allows for more than 40 options. This granularity of setup reduces the memory footprint of IIS 7.0, but more importantly, it reduces the security footprint as well. In IIS 6.0, a component such as CGI might never be used, but the code was still present in the core DLL. That means that a security exploit discovered in the CGI code will affect all IIS 6.0 installations, regardless of whether they use CGI. It also means that patches for the CGI code would need to be applied, even if you didn't run CGI.

The default installation of IIS 7.0 installs components needed for static HTML content, along with default documents, directory browsing, HTTP errors, and redirection. It also adds .NET extensibility for module extensions, as well as basic logging and tracing functions and request filtering (similar to the functionality provided by URLScan in previous versions), HTTP compression for static content, and the administration console. This means that, similar to IIS 6.0, a default installation can serve static content, with little other functionality.

Figure 1-3 shows the default installation options, enabling static content and very little else. Additional services, such as ASP.NET, can be installed either at installation or through configuration files. By leaving out services you don't need, the reduced amount of code provides for a reduced attack footprint for the overall installation. Installation options are covered in Chapter 4.

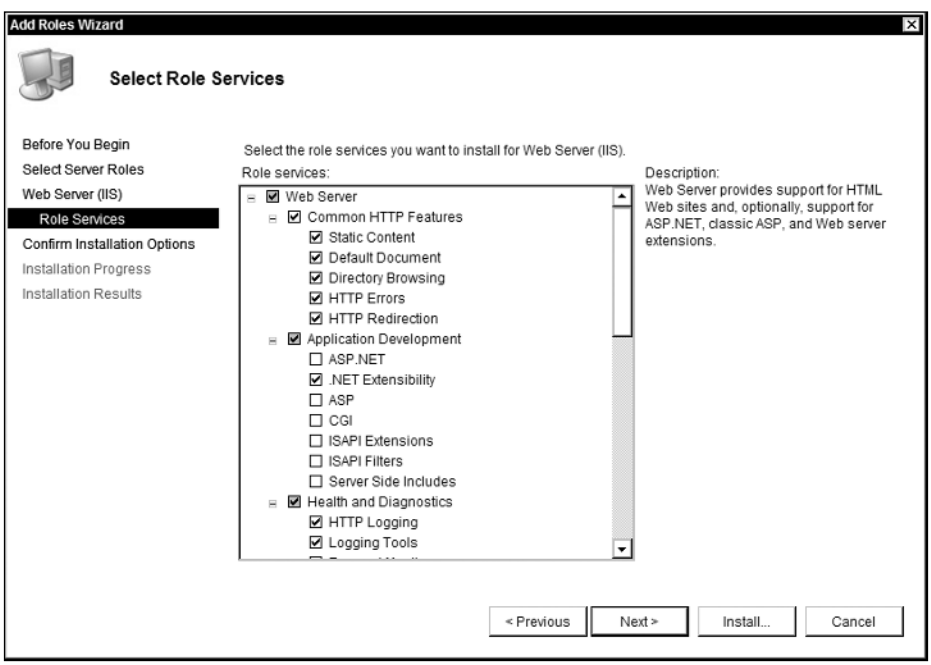

Figure 1-3

Management Delegation

Management of IIS in previous versions meant either granting local administrator privileges to the user or working through WMI and ADSI options to directly manage the site configurations. The only other option was for developers to work through the IIS administrators to change configurations — an option that could often be frustrating for both administrators and programmers. IIS 7.0 changes this through delegation of administration permissions at the server, site, and application levels.

In IIS 7.0, configuration options can be delegated in a very granular fashion. By default, most IIS settings are locked down and cannot be configured below the `applicationHost.config` file. You will see settings similar to this in the default file:

```
<sectionGroup name="system.applicationHost" type="…">
   <section name="applicationPools" overrideModeDefault="Deny" />
</sectionGroup>
```

To allow configuration delegation for a specific site, you would add a `<location>` element for that site, allowing the configuration files for the site to override the default settings in the `applicationHost.config` file. The code would be similar to

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

In a default installation, all IIS features are locked down except for HTTP, HTTP redirects, default documents, and directory browsing. All ASP.NET configurations are unlocked by default. In addition to delegations allowed within the configuration files, the configuration files themselves can be controlled through NTFS permissions. By setting ACLs on the files, an administrator can prevent unauthorized access to the files.

For an even more granular locking of specific elements in IIS, you can use attribute locking. Using `overrideMode`, an administrator can allow specific sites to be managed through configuration files by a developer. Attribute locking can be used to lock a specific attribute or element of the configuration while using `overrideMode="Allow"` on a web site. Developers can still override configurations at a local level, but the administrator maintains control of attributes they don't want changed. For example, to allow a developer to configure IIS options except for Windows authentication for the site MySite, you could use the following code in your `applicationHost.config` file to force the values required for Windows authentication:

```
<location path="MySite" overrideMode="Allow">
  <system.webServer>
    <security>
      <authentication>
        <windowsAuthentication enabled="true" lockAttributes="enabled">
          <providers>
            <add value="Negotiate" />
            <add value="NTLM" />
          </providers>
        </windowsAuthentication>
      </authentication>
```

```
      </security>
    </system.webServer>
  </location>
```

To allow the same developer on the same site to enable or disable Windows authentication but not to change the `providers` element, you could use

```
<location path="MySite" overrideMode="Allow">
  <system.webServer>
    <security>
      <authentication>
        <windowsAuthentication enabled="true" lockElements="providers">
          <providers>
            <add value="Negotiate" />
            <add value="NTLM" />
          </providers>
        </windowsAuthentication>
      </authentication>
    </security>
  </system.webServer>
</location>
```

Feature delegation extends to the GUI administration tool as well. At the server level, for example, you can configure which features can be changed by lower-level administrators using the Administration tool. You can configure administrators for any level in the Administration tool, and those administrators have access to features at or below their level. For example, server administrators can configure any site, whereas a site administrator can configure only features within that site.

Delegation of management functions is something administrators should consider carefully when planning an IIS 7.0 deployment in their organization. In Chapter 3, we discuss planning deployments. Chapter 6 covers using the `applicationHost.config` *file. Chapter 9 describes administration delegation.*

Unified Authentication and Authorization

In IIS 7.0, the authentication and authorization process merges the traditional IIS authentication options with ASP.NET options. This allows administrators and developers to use ASP.NET authentication across all files, folders, and applications in a site.

In IIS 6.0 and previous versions, controlling access to an Adobe Acrobat (PDF) file was difficult through ASP.NET authentication schemes. You would need to enable Windows authentication or basic authentication on the web site, folder, or file and create a Windows account to have access to the file. Then you would need to require the user to provide valid credentials for that Windows account, even if he or she already had logged into your ASP.NET application, to be able to access that PDF file. The alternative was to use impersonation in ASP.NET to access the file using the ASP.NET process account — all to prevent someone from opening the PDF file by pasting the direct URL into their browser. Options involving streaming the content from a protected location were just as cumbersome, and redirecting files to be processed by the ASP.NET DLL was even more problematic.

In IIS 7.0, using ASP.NET authentication no longer requires the file to be processed as an ASPX extension; thus file extensions of all types can be secured with Forms authentication or any other ASP.NET method. This reduces the requirement for Windows Client Access Licenses (CALs) to provide access

control, which was prohibitive in an Internet environment. It also allows, with the extensibility of the pipeline components, developers to create their own authentication schemes and easily apply them to any file or folder on the server.

Using ASP.NET Forms authentication for content other than ASP.NET is covered in Chapter 14, "Authentication and Authorization."

Request Filtering

IIS 7.0 includes request filtering as a standard function. While some of this ability was included in the unsupported URLScan tool released for IIS 5.0, request filtering takes this concept even further with hidden namespaces, where a particular section of a URL can be hidden and not served. Making the transition from using URLScan to request filtering is easy.

For example, in URLScan you could control serving specific file extensions using the `AllowExtension` or `DenyExtension` configurations. Request filtering uses the same allow or deny concept. For example, to allow all files to be served except for Microsoft Word files with a .DOC extension, you could use

```
<configuration>
   <system.webServer>
      <security>
         <requestFiltering>
            <fileExtensions allowUnlisted="true" >
               <add fileExtension=".doc" allowed="false" />
            </fileExtensions>
         </requestFiltering>
      </security>
   </system.webServer>
</configuration>
```

To allow only .ASPX files in a request, you could use

```
<configuration>
   <system.webServer>
      <security>
         <requestFiltering>
            <fileExtensions allowUnlisted="false" >
               <add fileExtension=".aspx" allowed="true" />
            </fileExtensions>
         </requestFiltering>
      </security>
   </system.webServer>
</configuration>
```

Denying access to a folder such as the BIN folder so that your DLLs could not be directly requested is handled by a new option called `hiddenNamespaces`. In URLScan, you could deny a URL sequence, so "BIN" could not appear in a URL, but that would affect requests for both `www.domain1.com/bin` and `www.domain1.com/binder/legalfiles`. With request filtering, you can hide the BIN folder by using

```
<configuration>
   <system.webServer>
      <security>
```

```
            <requestFiltering>
               <hiddenNamespaces>
                  <add hiddenDirectory="BIN" />
               </hiddenNamespaces>
            </requestFiltering>
         </security>
      </system.webServer>
   </configuration>
```

Using `allow` or `deny` in request filtering doesn't override any MIME-type settings or other security; it simply evaluates the HTTP request and allows it to be processed or rejects it based on filtering rules. If your Word documents were protected from being served by ACLs, they would be denied even without using request filtering, except the request would have to be processed to the step where access is denied instead of returning a failed result code immediately. The IIS result codes have been modified to indicate whether a request has been denied by request filtering.

Remote Management

Whilst IIS could be remotely managed in previous versions using the IIS Manager over RPC, this wasn't firewall friendly. A HTML based management option also existed, however this didn't allow management of all IIS features. In both cases, users were required to be in the local Administrators group on the machine.

IIS 7.0 introduces a new remote Management Service that permits the IIS Manager tool to administer remote IIS 7.0 installations over HTTPS. By utilizing the new delegation features in IIS 7.0, remote users can be given access to the entire server, a single website or even just a single web application. Additionally, features that have not been delegated will not be visible to the end user when connecting remotely.

Lastly, the Remote Management service introduces the concept of IIS Users. These user accounts do not exist outside of IIS. An administrator can choose to permit either Windows users, or IIS users, access to administer IIS remotely. IIS Users do not consume Windows client access licenses (CALs), nor do they have any permissions outside IIS itself, so are a cheaper and more secure option for permitting external IIS administration.

Although many security administrators will wisely insist on using a VPN for access from a public network, the remote Management Service is useful in hosting scenarios where a company has many external customers that you do not wish to allow access to the internal network. The remote Management Service is covered in Chapter 6, "Web-Site Administration."

IIS Administration Tools

IIS 7.0 uses a new IIS Manager that brings all the IIS and ASP.NET configurations into one management location. IIS 7.0 also has a full-functioned command-line tool for configuration, `AppCmd.exe`, as well as an ASP.NET namespace, `Microsoft.Web.Administration`, for management of all IIS functions through ASP.NET managed code. In addition, not only is there still WMI management functionality, but the management API has also been extended to allow complete control of all IIS features.

IIS Manager

The new IIS Manager for IIS 7.0, shown in Figure 1-4, combines all management functions for both IIS and ASP.NET in one location. Administrators will find the tool much easier to navigate than the MMC from previous IIS versions. Developers will find that they can manage individual sites and applications without needing local administrator access to the server. The IIS Manager is also extensible through the addition of modules.

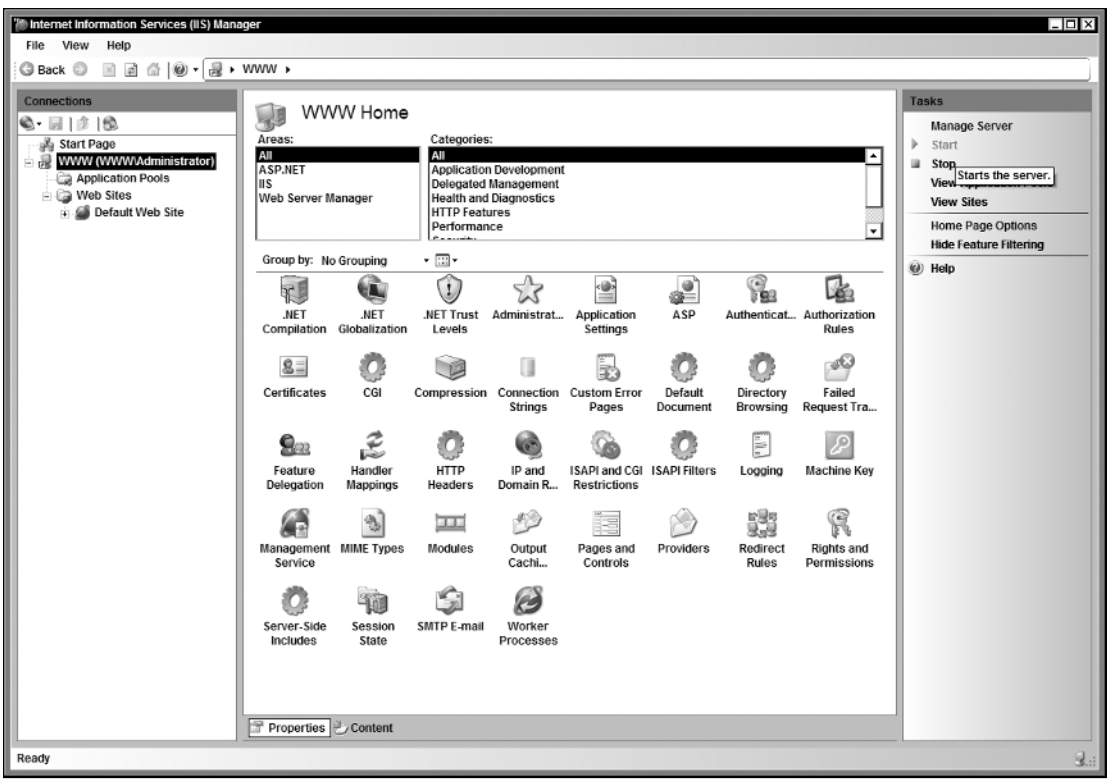

Figure 1-4

When you first open the IIS Manager, you will find that the interface is based on navigation and task. Each task that appears in the task pane is related to the point where you are in the navigation pane. Tasks that are not available at a specific navigation level are not displayed, and the task affects that navigation level and below. Additional sorting and selection options enable you to display only relevant tasks for the job at hand.

Administrators will also enjoy the delegation capabilities of the IIS Manager. You can have different administrators for specific applications within a site, as well as a site administrator, and each can configure functionality at or below their delegation level. The capability to delegate administration tasks allows non-administrators to administer web sites and/or applications, maintaining Windows security levels on the server itself. As shown in Figure 1-5, administration can be delegated to Windows accounts or IIS accounts, at any level.

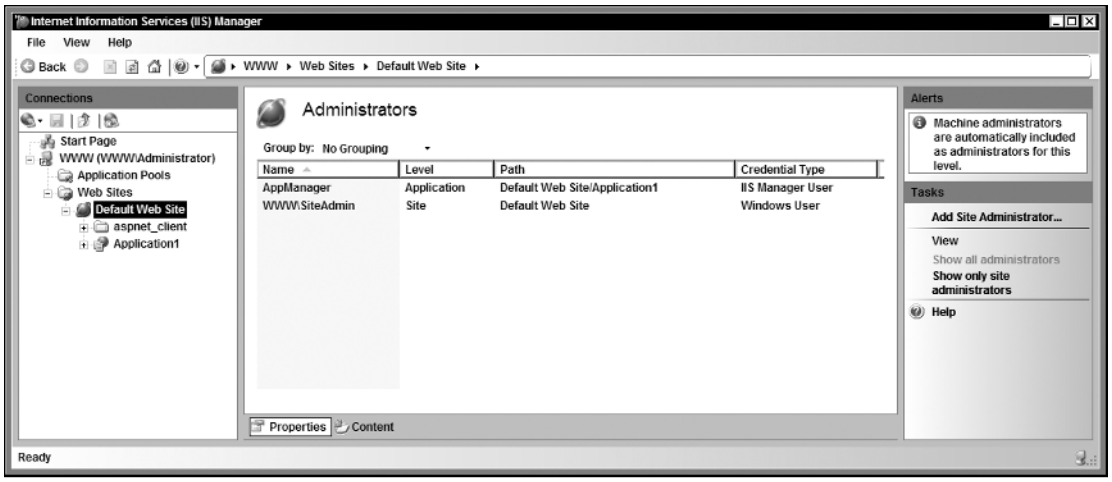

Figure 1-5

AppCmd.exe Command-Line Utility

IIS 7.0 introduces a new command-line utility, `AppCmd.exe`, which replaces the functionality provided by the various VBScript command-line utilities included with previous versions. `AppCmd.exe` also expands command-line control to all IIS configuration functions.

Creating a new web site from the command line is as easy as

```
appcmd.exe add site /name:MyNewSite /id:999 /bindings:"http/*:80:"
/physicalPath:"C:\inetpub\MyNewSite"
```

This single command line is shorter than the code involved in using the ASP.NET management namespace or WMI (each of which is described in the following two sections, respectively). The command line is even quicker than using an editor to enter the new web site directly into the `applicationHost.config` file.

One troublesome function in previous versions of IIS was obtaining a valid backup of the configuration of a single site. Exporting values from the registry or metabase, using Metabase Explorer or other tools, was possible but messy. And importing that backup into another server was all but impossible. `AppCmd.exe` makes a configuration backup a simple command line:

```
appcmd.exe add backup MyWebSiteBackup
```

Restoring that backup is as easy as

```
appcmd.exe restore backup MyWebSiteBackup
```

Before you make any configuration changes, you should run a quick backup using `AppCmd.exe`. Every administrator has at least one horror story about a long weekend spent restoring a configuration just because a simple change turned out to be not so simple.

ASP.NET Management Namespace

IIS 7.0 may be configured through the new ASP.NET namespace, `Microsoft.Web.Administration`, which is used for administration of IIS web sites and servers. This namespace is an addition to the ASP.NET framework that is installed with IIS 7.0.

An example of using the `Microsoft.Web.Administration` namespace would be the creation of a new web site. The following C# code sample creates a new site named *My New Site* with the root at c:\inetpub\MyNewSite, running HTTP on port 80:

```
using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.Web.Administration;
namespace MSWebAdmin_Application
{
   class Program
   {
      static void Main(string[] args)
      {
         ServerManager serverManager = new ServerManager();
         serverManager.Sites.Add("MyNewSite", "http", ":80:",
            "c:\\inetpub\\MyNewSite");
         serverManager.Sites["My New Site"].ServerAutoStart = true;
         serverManager.Update();
      }
   }
}
```

This would be the same as editing the `applicationHost.config` file with

```
<site name="My New Site" id="999" serverAutoStart="true">
   <application path="/">
      <virtualDirectory path="/" physicalPath="c:\inetpub\MyNewSite" />
   </application>
   <bindings>
      <binding protocol="http" bindingInformation=":80:" />
   </bindings>
</site>
```

More details and examples of using the `Microsoft.Web.Administration` namespace appear throughout the book. You can find the full documentation of the class at `http://msdn2.microsoft.com/en-us/library/microsoft.web.administration.aspx`.

Windows Management Instrumentation

The classic Windows Management Instrumentation (WMI) is still available in IIS 7.0. All your previous WMI scripts will work out of the box, and the API has been extended to include all features in IIS 7.0.

The example used in configuring a new web site through the `Microsoft.Web.Administration` namespace would look something like this using WMI:

```
Set oService = GetObject("winmgmts:root\WebAdministration")
```

```
Set oBinding = oService.Get("BindingElement").SpawnInstance_
   oBinding.BindingInformation = "*:80:"
   oBinding.Protocol = "http"
oService.Get("Site").Create
   "MyNewSite", array(oBinding), "C:\inetpub\MyNewSite"
oService.Get("Application").Create _
   "/", "MyNewSite", "C:\inetpub\MyNewSite"
```

The WMI provider for IIS 7.0 must be specifically installed, and Microsoft provides a set of WMI tools that can be downloaded from Microsoft.com.

Diagnostics

IIS 7.0 makes diagnostic tracing and server state management easy. Okay, it really makes it possible, since the diagnostic functions in IIS 7.0 didn't exist in previous versions. Run-time status can now be determined through a new set of APIs that expose the run-time state of sites, applications, and application pools, as well as allow for control of those states, through both WMI and managed code. The new Request Tracing module allows for tracing any request through the pipeline to the point of exit or failure, and provides a logging function for those traces.

Run-Time State and Control API

The Run-Time State and Control API allows for the determination of a point-in-time status of the server, site, or application, and the requests being processed. The API exposes the `HttpRequest` object in the worker process, as well as the application domain, and allows better tracing of a hung process and its cause. Within the `HttpRequest`, you can pull the pipeline state and current module the request is in, as well as details about the request such as the host name and IP address, client IP address, and URL requested. Runtime state information about running application pools, currently processing requests, and other exposed information is available from within the IIS Manager GUI, through the command line appcmd.exe tool and also programmatically through WMI and .NET management classes.

Request Tracing

Using the request tracing module, you can configure logging and tracing of any type of content or result code. Like most IIS settings, request tracing can be configured at the server, site, or application level.

Configuring request tracing is a simple task. First, you define a trace condition, such as File Not Found (result code 404) errors. Conditions can be based on result codes or the time taken for the request, or both. For example, a result code of 200 would only log a trace for a successful request. Once the trace condition is defined, you choose which trace provider to use and then begin logging traces that meet these conditions. Traces show in the log, where each step of the request is time stamped and you can see exactly what happened at each point in the request pipeline, as shown in Figure 1-6. Chapter 20, "Diagnostics and Troubleshooting," contains more information on request tracing.

Compatibility

IIS 7.0 has maintained compatibility with IIS 6.0 for easy migration of existing web sites. All existing ISAPI filters, classic ASP and ASP.NET applications, and ADSI or WMI scripts will work in exactly the same way they did with IIS 6.0. This is handled through two primary means. Firstly, IIS 7.0 has a Classic mode application pool setting that allows an application pool to function in the same way that it did in

IIS 6.0, in case your application is not able to run in the new IIS 7.0 integrated pipeline mode. Secondly, an optional IIS 6.0 Metabase Compatibility module can be installed for those applications or scripts that query the IIS metabase. Installing this module installs an ABO mapper that transparently redirects calls to metabase properties to the corresponding section in the new applicationHost.config file. All existing scripts and code (whether they read or write to the metabase) should continue to workThis mapper does not allow any new IIS 7.0 functionality to be configured, and scripts using it are limited to IIS 6.0 features. For example, an existing script could create a new web site but could not configure failed request tracing for that site, because there are no metabase properties related to request tracing. ASP.NET configurations are also unavailable through this wrapper.

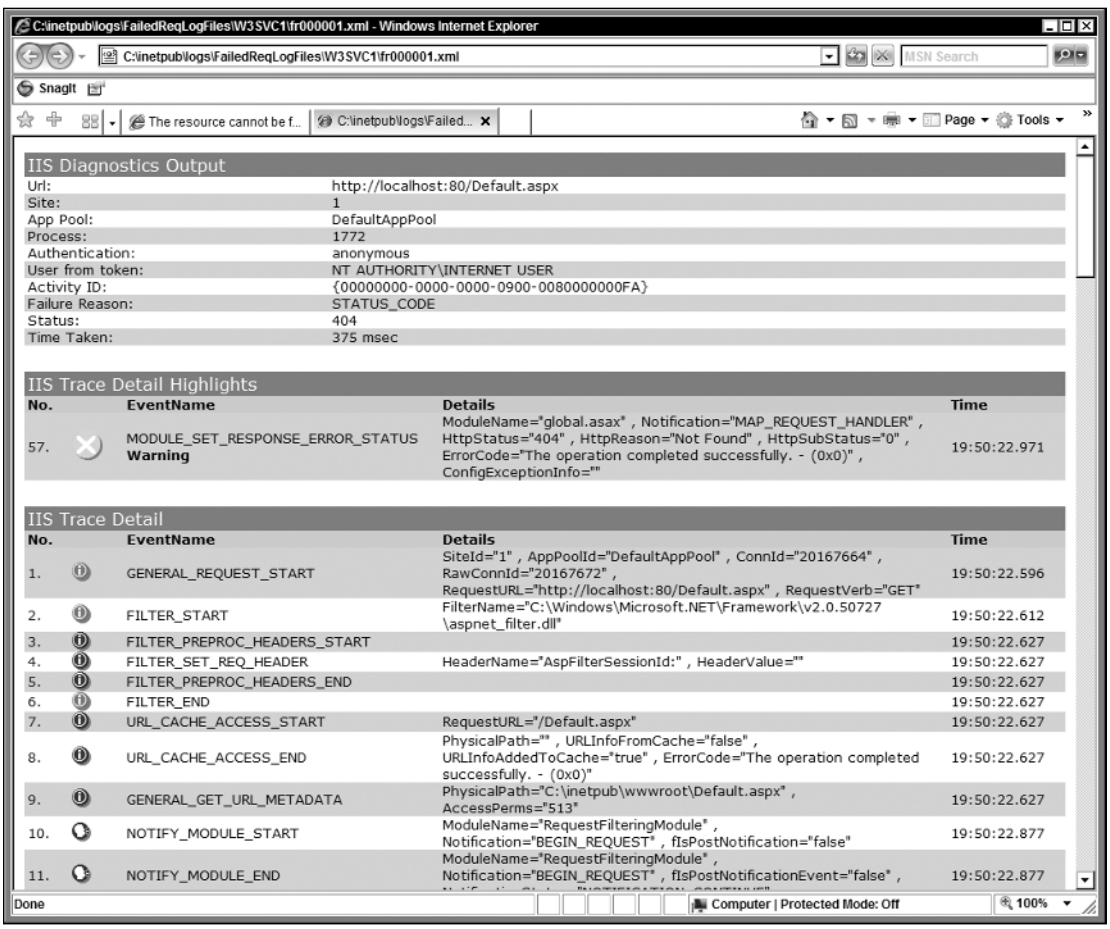

Figure 1-6

There are two possible compatibility issues you may face in migrating sites from IIS 6.0. The first has to do with the new security model for Vista/Longhorn. Because the security model is one of least privileges, if you have altered the default settings or accounts for IIS 6.0 on Windows Server 2003, you may need to correct permission errors that show up in IIS 7.0. Depending on whether the site and/or applica-

tion are on a server that was upgraded to Longhorn or have been migrated to a new installation of Longhorn, you may need to create new accounts and assign the correct permissions.

The second caveat that will catch you on transfers to a new IIS installation is the modularity of IIS 7.0. On a default install of IIS 7.0, components such as classic ASP, ASP.NET, and ISAPI filters and extensions are not installed; thus these functions will not work if you simply copy the site contents and code over to a new default IIS 7.0 installation. You must install components in IIS 7.0 that were used in your site under IIS 6.0 for the transfer to work seamlessly. Upgrades of existing IIS 6.0 setups will retain all their previous functionality but will not receive any new functionality unless specifically configured.

Additional Features

As in previous versions, IIS 7.0 includes FTP and SMTP services. SMTP remains unchanged from Windows Server 2003 and IIS 6.0, as does the version of FTP that ships with Windows Server 2008. A new FTP server is available as a free download from Microsoft's web site at `www.iis.net`, and this version incorporates many of the requested changes from customers as well as tight integration with web sites to simplify publishing through FT. Windows Server 2008 drops the POP3 service that was introduced in Windows Server 2003. Both FTP and SMTP are covered in detail in Chapter 10.

FTP

Windows Vista shipped with exactly the same FTP code and functions found in Windows Server 2003 and IIS 6.0, and Windows Server 2008 ships with the same code as well. A new FTP server, shipped as a free download from `www.iis.net`, includes secure FTP using SSL certificates. This has been one of the primary reasons for using third-party FTP servers. In addition, the new version of FTP for Windows Server 2008 is integrated with the IIS 7.0 management functions, including extensibility of the authentication process. This means that FTP can use ASP.NET authentication, including membership and roles features, and will not require Windows CALs. Both versions of FTP are covered in detail in Chapter 10.

SMTP

SMTP is still available on Windows Server 2008, as it was on Windows Server 2003, without the need to purchase Microsoft Exchange Server. Unchanged from the Windows Server 2003 implementation, SMTP code is actually developed and owned by the Windows Exchange Server development team. The SMTP service in Windows Server 2008 is not meant to be a full-featured implementation, but rather a simplified service that provides minimum functionality without the need for additional services. Most professional users of IIS will want to install another mail server product, such as Microsoft's Exchange Server.

That doesn't mean that SMTP in Windows Server 2008 is a lightweight product. It is still functional for sending mail from applications on IIS 7.0, and it is a fully compliant implementation of SMTP that functions well in an Internet environment. While not having the configurability of Microsoft's Exchange Server, it will still function with multiple virtual servers and serve multiple SMTP domains while providing for security through relay permissions and IP restrictions as well as Windows login account access.

Windows Server 2008 no longer provides a POP3 server, and no IMAP functionality is available without additional products installed. Chapter 10 goes into more detail on SMTP installation and configuration.

Summary

IIS 7.0 is an evolution of previous IIS versions, building on their strengths while overcoming their weaknesses. Microsoft has listened to user comments on previous versions and responded positively, to make IIS 7.0 the most robust, configurable, and secure version of their web server. In addition, Microsoft has further enhanced the tie between IIS and ASP.NET, making IIS 7.0 a fully functional application server with entirely integrated processing and configuration of both sides.

Programmers will see the improvements in IIS 7.0 as making their job easier and more manageable. They will no longer need to justify web server changes that might affect numerous sites just to adapt to their application. True XCopy deployment of applications and configurations will allow them to spend less time on deployment schemes that need to work around server configurations. Senior programmers on a site will be able to delegate appropriate permissions for configuration changes to junior programmers, just as administrators will delegate permission to developers.

Administrators will find that the delegation in IIS 7.0 greatly assists in lowering the administrative effort required to maintain an IIS server. Administrators will be able to granularly control their servers and site administration functions while allowing developers to control the areas they need to. Security concerns about opening servers to development staff are a thing of the past, and auditors will be pleased with the control available. The reduction in security exposure provided by the modular installation, as well as the ability to provide unified authentication across an entire site, will further help to lock down an already secure system. Administrators will also enjoy the simplified and unified management tools.

Both administrators and developers will appreciate IIS 7.0's tracing and diagnostic capabilities. The ability to see into a request will ease troubleshooting failed requests as well as allow better tuning for performance even in completed requests. The modularity of IIS will also appeal to both camps, especially the ability to write custom modules that sit in the request pipeline and affect all requests, whether for a custom authentication system or simply to add a copyright notice to all images or content served.

After reviewing this chapter, you should have a good idea of the changes and new features in IIS 7.0. The following chapters will take you through installing and configuring the web server and environment. If you have a particular interest, feel free to skip to the chapter or chapters covering it. Otherwise, let's take an in-depth look at the architecture of IIS 7.0, followed by planning your installation and installing IIS 7.0.