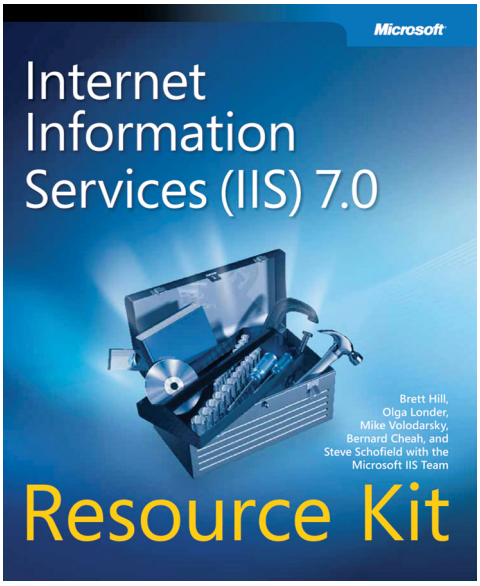

Internet Information Services (IIS) 7.0 Resource Kit

Mike Volodarsky, Olga Londer, Brett Hill, Bernard Cheah, and Steve Schofield with the Microsoft IIS Team

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

Microsoft Press

978-0-7356-2441-2

Table of Contents

Part I
Foundation

Chapter 3
Understanding the Modular Foundation

What does *modular core* mean to Internet Information Services (IIS) 7.0? How does it make IIS 7.0 the most powerful Microsoft Web server ever? And what are the built-in modules shipped with IIS 7.0? No worries—by the end of this chapter, you will be able to answer all these questions and have a clear understanding of the new design concept behind IIS 7.0. You will take a look at the idea of componentized design in IIS 7.0, the intentions behind the revamped architecture, and the advantages of the design, as well as detailed information about the built-in modules that ship with IIS 7.0.

Concepts

One of the core changes for IIS 7.0 is its component-based architecture, which incorporates lessons learned from IIS 6.0 and feedback from customers. IIS 7.0 debuts with a completely overhauled and redesigned architecture; the Web server core is now broken down into discrete components called *modules*. For the first time, as a Web administrator you have the power to custom build an IIS server according to your requirements. You can easily add built-in modules whenever they are needed or, even better, add or replace functionality with modules of your own design, produced commercially or provided by the developer community on IIS.net. In this way, the modular engine will allow you to achieve exactly the functionality you want from the Web server, and at the same time provide flexibility so you can remove unwanted modules to further secure and better lock down the Web server.

Although the main modularity point in IIS 7.0 is the Web server itself, features throughout the entire platform are implemented as modules. The administration stack for example is modular. For detailed information about extensibility of the IIS 7.0 Web server and the administration stack, see Chapter 11, "Managing Web Server Modules" and Chapter 12, "Managing Configuration Extensions".

The Ideas

A module resembles a brick in a child's LEGO™ toy set, which comes with bricks in many different colors and shapes. When combined with additional bricks you can assemble many different models in a variety of shapes. IIS 7.0 uses the same idea in the design of its framework foundation. By using modules as the building blocks, this pluggable architecture combined with the flexible configuration system and an extensible User Interface (UI) make it possible to add or remove any capability to craft a server that fits the specific needs of your organization. This new and open design is revolutionary for Microsoft and opens new doors for the web platform.

How It Works: The Modular Design

IIS 7.0 ships with many different modules. Each module is a component (not in the Component Object Model (COM) sense) that provides services to the Web server's HyperText Transfer Protocol (HTTP) request-processing pipeline. For example, the StaticFileModule is the module that handles all static content such as HyperText Markup Language (HTML) pages, image files, and so on. There are other modules that provide capabilities for dynamic compression, basic authentication, and the other features you typically associate with IIS. Modules are discretely managed in IIS 7.0. They can easily be added to or removed from the core engine via the new configuration system. Internally, the IIS Web server core provides the request processing pipeline for modules to execute. It also provides request processing services, whereby modules registered in the processing pipeline are invoked for processing requests based on registered event notifications. As an administrator, you cannot control which events the modules are coded to use. This is done in the code within the module. However, you have the ability to control which modules are loaded globally, and you can even control which modules are loaded for a specific site or application. For details about how to control module loading, see the Chapter 11, "Managing Web Server Modules".

Each time the IIS 7.0 worker process starts, it reads the server configuration file, and loads all globally listed modules. Application modules are loaded upon the first request to the application. It is the modular design and configuration system that makes it easy for you to plug in, remove, and replace modules in the request pipeline, offering full extensibility to the IIS 7.0 Web server.

Types of Modules

IIS 7.0 ships with approximately 40 modules, including security-related authentication modules and modules for content compression. Modules build up the feature sets of the Web server, and the Web application is made of up many modules servicing the requests. In terms of roles, modules can be categorized as providing either *request services* such as compression and authentication or *request handling* such as delivering static files, ASP.NET filtering, and so on. Regardless of their roles, modules are the key ingredients to IIS 7.0. In terms of how they are coded, there are two types of modules in IIS 7.0:

Managed modules

- A managed module is a .NET Framework component based on the ASP.NET extensibility model. With the IIS 7.0 integrated processing architecture, ASP.NET application services are no longer restricted to requests for .ASPX pages or other content mapped to ASP.NET. The managed modules are plugged in directly to the Web server's request processing pipeline, making them as powerful as the modules built using IIS 7.0's native extensibility layer. Now, you can also provide ASP.NET module services across all requests; however, this requires running in the integrated process model with the ManagedEngine module installed. The ManagedEngine

component is a special module provided in IIS 7.0 that provides the .NET integration into the request processing pipeline. Managed modules are loaded globally only when the application pool is marked as "Integrated". For more information about the new integrated pipeline processing mode, see Chapter 11, "Managing Web Server Modules".

Native modules

- A native module is a Microsoft Windows Dynamic Link Library (DLL) typically written in C++ that provides request processing services. In IIS 7.0, a new set of native server (C++) Application Programming Interfaces (APIs) replaced the Internet Server API (ISAPI) filters and extension APIs provided by earlier versions of IIS. These new APIs are developed in an object-oriented model and are equipped with more powerful interfaces that allow you more control when it comes to processing requests and handling responses. Additional integration between Microsoft Visual Studio development suites and IIS make developing IIS modules easier than ever. Developer's familiar with ISAPI and the new enhancements in native module APIs have been very positive about how much easier it is now to code using native code than in previous versions of IIS.

> **Note** For details on how to write native modules, see:
> *http://www.iis.net/articles/view.aspx/IIS7/Extending-IIS7/Building-Native-Modules/Develop-a-Native-C-C---Module-for-IIS7*

Native and managed modules are managed and configured the same way in IIS 7.0 with the exception of deployment of the modules. Native modules must be installed on the server and registered in the **<**`globalModules`**>** section of the applicationHost.config file before they can be enabled for application usage; managed modules are registered directly in an application's web.config file rather than installed. For more information about the deployment of modules, see Chapter 11, "Managing Web Server Modules".

Modules and Configuration

For modules to provide certain features or services to IIS 7.0, the modules must be registered in the configuration system. This section looks at the relationship between modules and various sections in the configuration file and provides a high-level overview of the module settings in the configuration store. For more information about the IIS 7.0 configuration system, which is based on Extended Markup Language (XML), see Chapter 4, "Understanding the Configuration System".

Inside the `<system.webServer>` section group of the applicationHost.config file (the main server configuration file), there are three different sections related to modules:

`<globalModules>`

- Configurable at the server level only, this section defines all native code modules that will provide services for requests. The module declaration in the configuration section also specifies the related DLL file that provides the module's features. All native modules must be defined or registered in this section before they can be turned on or enabled for application usage as defined in the `<modules>` section.

```
// Example of <globalModules> configuration section
<globalModules>
...
<add name="StaticCompressionModule" image="%windir%\...\compstat.dll" />
<add name="DefaultDocumentModule" image="%windir%\...\defdoc.dll" />
<add name="DirectoryListingModule" image="%windir%\...\dirlist.dll" />
...
</globalModules>
```

<handlers>

- Configurable at the server level, the application level, and the Uniform Resource Locater (URL) level, this section defines how requests are handled. It also maps handlers based on the URL and HTTP verbs, specifying the appropriate module that supports the related handler. By parsing the handler mapping configuration, IIS determines which modules to call when a specific request comes in.

```
// Example of <handlers> configuration section
<handlers accessPolicy="Script, Read">
...
<add name="ASPClassic" path="*.asp" verb="GET,HEAD,POST"
  modules="IsapiModule" scriptProcessor="...\asp.dll" resourceType="File" />
<add name="SecurityCertificate" path="*.cer" verb="GET,HEAD,POST"
  modules="IsapiModule" scriptProcessor="...\asp.dll" resourceType="File" />
<add name="SSINC-stm" path="*.stm" verb="GET,POST"
  modules="ServerSideIncludeModule" resourceType="File" />
...
</handlers>
```

<modules>

- Configurable at the server level and the application level, this section defines modules enabled for the application. Although native modules are registered in the `<globalModules>` section, native modules must be enabled in the `<modules>` section before they can provide their services for requests to applications. Managed code modules however can be added directly to the `<modules>` section. For example, you can add a custom managed basic authentication module to an application's web.config file.

```
// Example of <modules> configuration section
<modules>
...
<add name="BasicAuthenticationModule" />
<add name="WindowsAuthenticationModule" />
<add name="OutputCache" type="System.Web.Caching.OutputCacheModule"
  preCondition="managedHandler" />
<add name="Session" type="System.Web.SessionState.SessionStateModule"
  preCondition="managedHandler" />
...
</modules>
```

Key Benefits

The modular architecture in IIS 7.0 offers many advantages compared with previous versions of IIS. This section outlines the benefits derived from the componentized design of IIS 7.0. It also provides scenarios illustrating how a Web administrator can take advantage of these benefits while building a robust Web server.

Security

Security is of the utmost concern when it comes to today's web applications. IIS 6.0 is not installed by default except in the Windows Server 2003 Web Server edition. The IIS 6.0 default installation serves static content only; all other functionality is disabled. IIS 7.0 reflects the Web server's modular nature, allowing the user to install only the modules that they require for their application. Binaries that comprise the other features are not installed, but instead are kept in a protected OS installation cache. This means that you will not be prompted for a CD or asked to point to a source location when installing new updates or adding features–yet, the binaries that you are not using are not loaded by the IIS worker processes, rather they are quarantined such that they cannot be accessed. When security updates from Microsoft are applied, the features that have not been installed will be fully updated in the installation cache. This can eliminate the need to reapply service packs when installing new features later.

From the security perspective, the modular design brings several key advantages including:

Minimized attack surface

- By giving you the power to install only those components that are needed, IIS 7.0 directly minimizes the areas of possible attack. The attack points are limited to the installed components because the binaries exist only for the installed components. Because only exposed areas or installed components can be subject to potential exploits, this is the best defense. For example, with the IIS 7.0 default installation, there are about 10 components installed to support internal IIS logging and management as well as serving static content requests. Technically speaking, these are the only surfaces that are exposed for potential attack.

Reduced maintenance overhead

- Modular design not only provides new flexibility when adding, removing, and even replacing components, it also provides a new maintenance experience through opt-in patching. You need apply fixes or patches only to required or installed components; unused components or modules that have not been installed do not require *immediate* attention and no downtime is required when patching components that are not installed. It also means that fewer administrative tasks are needed for routine maintenance and upgrades. For example, if an IIS 7.0 server uses Windows authentication only for its applications, only Windows authentication module patches are applicable to the server. If a patch is distributed to protect the Basic authentication module from a known exploit, and a patch is not immediately required because the Basic authentication module is not installed. Note however that Microsoft recommends that you apply all patches to ensure that modules and features you are not using will be current in the event they are installed later.

> **Important** Microsoft recommends that you apply all patches to the server. When patching components that aren't in use, the server doesn't have to experience any downtime. If the components are eventually installed, the latest versions of their binaries will automatically be used, and there is no need to re-apply any patches.

Unified Security Model

- IIS 7.0 is now better integrated with ASP.NET. Having both IIS 7.0 native modules and ASP.NET managed modules running in the same request pipeline yields many benefits including unifying the configuration system and security models for both IIS and ASP.NET. From the security perspective, ASP.NET advanced security services can be plugged in directly to the IIS main request processing pipeline and used together with the security features offered by IIS. In short, with IIS 7.0 it is now possible to configure ASP.NET security services for non ASP.NET requests. For example, with earlier versions of IIS if an application consists of both PHP and ASP.NET resources, ASP.NET Forms authentication can be applied to only ASP.NET resources. With the IIS 7.0 integrated process model, it is now possible to have Forms authentication for PHP, ASP.NET, and any other type of resources.

Direct from the Source: The Most Secure Web Server in the World

The first time we presented IIS 7.0 to a large audience was also my first TechEd breakout session, hosted at TechEd 2005. My first demo showcased the componentization capabilities of IIS 7.0 by showing off what we jokingly called "the most secure Web Server in the world."

As part of the demo, I walked through how to edit the configuration in the applicationHost.config file, removing all of the modules and handler mappings. After saving the file, IIS would automatically pick up the changes and restart, loading absolutely no modules. After making a request to the default web site, I would swiftly get back an `empty 200` response (this configuration currently returns a `401 Unauthorized` error because no authentication modules are present). The server had no modules loaded, and therefore would perform virtually no processing of the request and return no content, thus truly becoming the "most secure Web Server in the world". After a pause, I commented that, while secure, this server was also fairly useless, and then I segued into adding back the functionality that I needed for my application.

I had done this demo before for internal audiences to much acclaim, but I will always remember the audience reaction during that TechEd session. The people in the audience went wild, some even breaking into a standing ovation. This was a resounding confirmation of our efforts to give administrators the ability to start from nothing, building up the server with an absolutely minimal set of features to produce a simple-to-manage Web Server with the least possible surface area.

Mike Volodarsky

IIS7 Core Server Program Manager

Performance

With its componentized architecture, IIS 7.0 provides very granular control when it comes to the Web server memory footprint. Modules are loaded into memory only if they are installed and enabled. By removing unnecessary IIS 7.0 features, fewer components are loaded in the processing pipeline—in other words, fewer steps are needed to fulfill incoming requests and, therefore, overall server performance improves. At the same time, by reducing memory usage for the IIS 7.0 server, more free memory space is available for the Web application and operating system. For example, in IIS 6.0, all authentication providers (Anonymous, Windows, Digest, and so on) are loaded in the worker process; in IIS 7.0, only the needed authentication modules are loaded and included in the request processing. For more details on removing modules you do not require, see Chapter 11, "Managing Web Server Modules".

Extensibility

In earlier versions of IIS, extending or adding IIS features is not easy because it can be done only through ISAPI programming with limited API support and limited access to information in the request processing pipeline. With the new modular-based engine and the tight integration between ASP.NET and IIS, extending IIS 7.0 is much easier. Not only are you able to decide which features to include in the Web server, you can extend your Web server by adding your own custom components to provide specific functionality. For example, you can develop an ASP.NET basic authentication module that uses the Membership service and a SQL Server user database in place of the built-in IIS Basic authentication feature that works only with Windows accounts. In short, you can build your own custom server to deliver the feature sets your applications require. You might, for example deploy a set of IIS 7.0 servers just for caching purposes, or you might deploy a custom module to perform a specific function in an application such as implementing your own ASP.NET application load balancing algorithm based on customer requirements. For more information on customizing modules in IIS 7.0, see Chapter 11, "Managing Web Server Modules".

Built-in Modules

Modules shipped with IIS 7.0 are grouped into different categories accordingly to the role of the services they provide. Table 3-1 highlights the different service categories and lists sample built-in modules within those categories. A complete list of modules is included in Appendix XM, "Module Listing".

Table 3-1 Module Categories

Category	Module
Application Development	CgiModule (%windir%\system32\inetsrv\cgi.dll) Facilitates support for Common Gateway Interface (CGI) programs
	FastCgiModule (%windir%\system32\inetsrv\iisfcgi.dll) Supports FastCGI, which provides a high-performance alternative to old-fashioned CGI-based programs
	System.Web.SessionState.SessionStateModule (ManagedEngine) Provides session state management, which enables storage of data specific to a single client within an application on the server.
Health and Diagnostics	FailedRequetsTracingModule (%windir%\system32\inetsrv\iisfreb.dll) More commonly known as Failed Request Event Buffering (FREB), this module supports tracing of failed requests. The definition and rules defining a failed request can be configured.
	RequestMonitorModule (%windir%\system32\inetsrv\iisreqs.dll) Implements the Run-time State and Control API (RSCA). RSCA allows its consumers to query run-time information such as currently executing requests, the start or stop state of a Web-site, or currently executing application domains.
HTTP Features	ProtocolSupportModule (%windir%\system32\inetsrv\protsup.dll) Implements custom and redirect response headers, handles HTTP TRACE and OPTIONS verbs, and supports keep-alive configuration.
Performance	TokenCacheModule (%windir%\system32\inetsrv\cachtokn.dll) Caches windows security tokens for password-based authentication schemes (anonymous authentication, basic authentication, and IIS client certificate authentication).
	System.Web.Caching.OutputCacheModule (ManagedEngine) Defines the output caching policies of an ASP.NET page or a user control contained in a page
Security	RequestFilteringModule (%windir%\system32\inetsrv\modrqflt.dll) Provides URLSCAN-like functionality in IIS 7.0 by implementing a powerful set of security rules to reject suspicious request at a very early stage.
	UrlAuthorizationModule (%windir%\system32\inetsrv\urlauthz.dll) Supports rules-based configurations for content authorization.
	System.Web.Security.FormsAuthenticationModule (ManagedEngine) Implements ASP.NET Forms authentication against requested resources.
Server Components	ConfigurationValidationModule (%windir%\system32\inetsrv\validcfg.dll) Responsible for verifying IIS 7.0 configuration systems, such as when an application is running in Integrated mode but has handlers or modules declared in the <system.web> section.
	ManagedEngine / ManagedEngine64 (webengine.dll) Managed Engine has a special place within all the other modules. It is responsible for integrating IIS with the ASP.NET runtime.

For more information regarding the module configuration store, module dependencies, and potential issues when a module is removed, see Appendix XM, "Module Listing".

Chapter Summary

The key features delivered by IIS 7.0 come from the modular design. This is the first time the Web administrator has full control over the IIS server. It is also the first version of IIS that is fully extensible. It provides a unified request processing model that integrates ASP.NET and IIS. Modules are fundamental building blocks to IIS 7.0 server. IIS 7.0 provides quite a few ways to manage modules (the basic unit of the IIS feature set) in order to implement efficient low-footprint Web servers optimized for a specific task. By choosing the right set of modules, you can enable a rich set of functionality on your server, or you can remove features you do not need to reduce the security surface area and improve performance. In Chapter 11: Managing Web Server Modules, you can learn more about the different types of modules IIS 7.0 supports, how they work, and learn how to properly deploy and manage them in the IIS environment.