ASP.NET Integration with IIS 7

* Author: Mike Volodarsky
* Published on December 05, 2007 by [IIS Team](http://forums.iis.net/members/IIS+Team.aspx)
* Updated on May 21, 2010 by [IIS Team](http://forums.iis.net/members/IIS+Team.aspx)

Introduction

Since its release, ASP.NET has been the platform of choice for developing Web applications on the Windows / IIS platform. ASP.NET 2.0 took Web application development to a new level, allowing developers to build more powerful applications faster than ever before.

IIS 7 takes ASP.NET further by integrating the ASP.NET runtime extensibility model with the core server. This allows developers to fully extend the IIS 7 server with the richness of ASP.NET 2.0 and the .NET Framework, instead of using the less capable IIS C++ APIs. Existing ASP.NET applications also immediately benefit from tighter integration by using existing ASP.NET features like Forms authentication, roles, and output caching for all content.

While IIS 7 provides the improved ASP.NET integration by default, there is a choice: IIS 7 supports both the new and the old ASP.NET integration modes that can be used side by side on the same server.

This article discusses the improvements that are introduced by the new ASP.NET integration mode, the architecture of the two modes, and describes how to select and configure the integration modes for ASP.NET applications.

ASP.NET Enhancements on IIS 7

Better ASP.NET integration in IIS 7 enhances existing applications and also allows new applications to take advantage of ASP.NET features in new ways:

* **ASP.NET services can be used for all content types**. In the past, ASP.NET functionality such as Forms authentication, roles, URL authorization, and output caching were only available to ASP.NET content types like ASPX pages. Static files, ASP pages, and other content types could not benefit from these services.

In IIS 7, all ASP.NET services are provided uniformly to all content. For example, you can protect all of your Web content, including images and ASP pages, with your existing ASP.NET 2.0 access control solution built on ASP.NET Forms authentication, membership and login controls.

* **Fully extend IIS with ASP.NET.** Previous versions of IIS frequently required server extensibility to be developed by using the native ISAPI filter or extension extensibility mode, due to the runtime limitations of ASP.NET.

IIS 7 allows ASP.NET modules to plug in directly into the server pipeline, with the same runtime fidelity as modules developed with the native (C++) server API. ASP.NET modules can execute in all runtime stages of the request-processing pipeline, and can be executed in any order with respect to native modules. The ASP.NET API is also expanded to allow more control over request processing than was previously possible.

* **Unified server runtime.** Tighter ASP.NET integration also unifies many of the features between IIS 7 and ASP.NET.

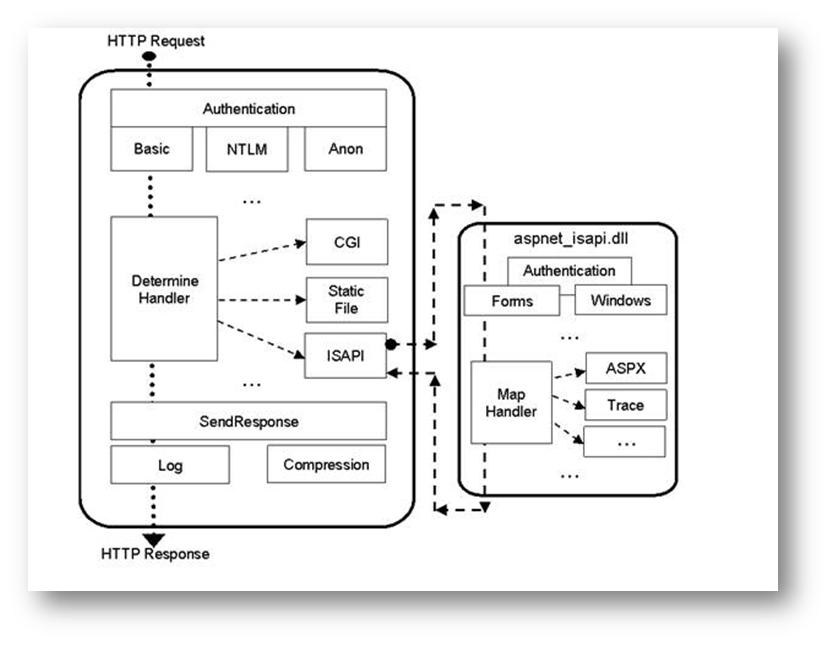
IIS 7 provides a unified configuration for IIS and ASP.NET modules and handlers. Many other features, including custom errors and tracing, have been unified to allow better management and cohesive application design.

ASP.NET Integration Architecture

In IIS 6.0 and previous releases, ASP.NET was implemented as an IIS ISAPI extension.

In these earlier releases, IIS processed a request to an ASP.NET content type and then forwarded that request to the ASP.NET ISAPI DLL, which hosted the ASP.NET request pipeline and page framework. Requests to non-ASP.NET content, such as ASP pages or static files, were processed by IIS or other ISAPI extensions and were not visible to ASP.NET.

The major limitation of this model was that services provided by ASP.NET modules and custom ASP.NET application code were not available to non-ASP.NET requests. In addition, ASP.NET modules were unable to affect certain parts of the IIS request processing that occurred before and after the ASP.NET execution path.

[](http://learn.iis.net/file.axd?i=151)

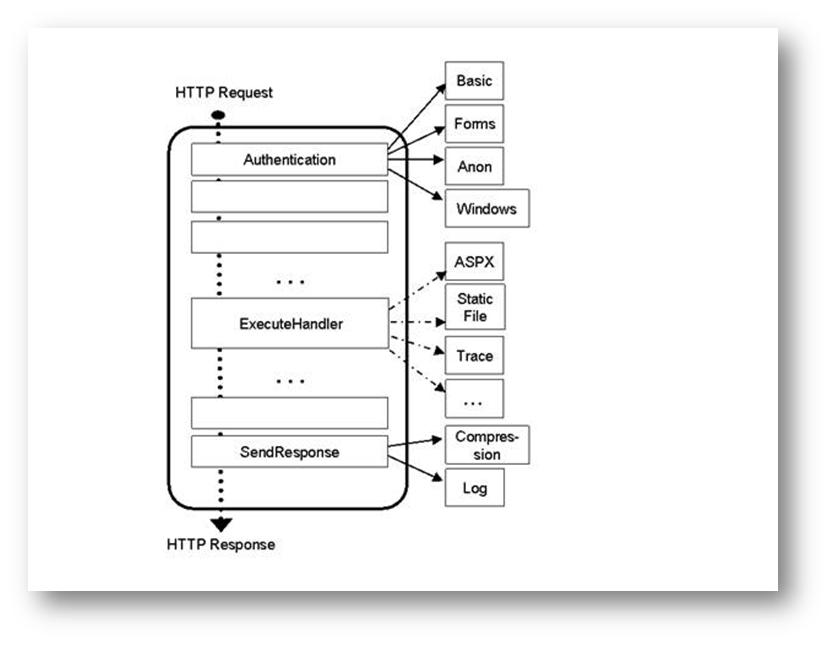
**Figure 1: IIS 6.0 & ASP.NET Pipelines**

In IIS 7, the ASP.NET request-processing pipeline overlays the IIS pipeline directly, essentially providing a wrapper over it instead of plugging into it.

IIS 7 processes requests that arrive for any content type, with both native IIS modules and ASP.NET modules providing request processing in all stages. This enables services that are provided by ASP.NET modules, such as Forms authentication or output cache, to be used for requests to ASP pages, PHP pages, static files, and so on.

The ability to plug in directly into the server pipeline allows ASP.NET modules to replace, run before, or run after any IIS 7 functionality. This enables, for example, a custom ASP.NET Basic authentication module that is written to use the Membership service and SQL Server user database to replace the built-in IIS Basic authentication feature that works only with Windows accounts.

In addition, the expanded ASP.NET APIs use direct integration to enable more request-processing tasks. For example, ASP.NET modules can modify request headers before other components process the request, by inserting an Accept-Language header before ASP applications execute, which forces localized content to be sent back to the client based on user preference.

[](http://learn.iis.net/file.axd?i=152)

**Figure 2: IIS 7 Integrated Mode**

Because of the runtime integration, IIS 7 and ASP.NET can use the same configuration to enable and order server modules, and to configure handler mappings. Other unified functionality includes tracing, custom errors, and output caching.

Compatibility

Most notably, the architecture maintains the ASP.NET runtime architecture and APIs, which allows existing ASP.NET applications and services to work upon installation. With a few modifications, the existing ASP.NET applications and services can take advantage of the new ASP.NET capabilities.

Likewise, developers can continue to write new applications to familiar ASP.NET APIs while they harness the benefits of the runtime integration.

IIS 7 continues to provide the Classic ASP.NET mode for ASP.NET applications that have specific compatibility requirements that are not met by the Integrated mode. Administrators can select the desired integration mode per application pool, which lets applications that use both the new and the Classic ASP.NET modes to function side-by-side on the same server.

Migrating ASP.NET Applications to IIS 7 Integrated Mode

On IIS 7, ASP.NET is configured to operate in the new Integrated mode by default. This lets your application take advantage of Integrated mode enhancements with minimal modifications.

Because of the configuration unification, some applications may require migration to operate properly in Integrated mode. By default, the server provides assistance with migration. It detects applications that require migration, and returns an error message requesting that you migrate the application.

The following configuration causes the migration error:

1. **The application Web.config file defines <httpModules> configuration.**  
   The application loads new ASP.NET modules or removes existing ones.   
   In Integrated mode, ASP.NET modules are specified with native modules in the unified <system.webServer>/<modules> configuration section.  
   The ASP.NET modules that are specified in the <system.web>/<httpModules> configuration section must be moved to the new configuration section to take effect. Subsequently, new ASP.NET modules must be added directly to the unified <modules> section.
2. **The application Web.config file defines <httpHandlers> configuration.**  
   The application uses custom handler mappings for some content types.   
   In Integrated mode, ASP.NET handler mappings must be specified in the unified <system.webServer>/<handlers> configuration section to take effect. Subsequently, new ASP.NET handler mappings must be added directly to the unified <handlers> section.  
   This section replaces both the ASP.NET <httpHandlers> configuration and the IIS 7 scriptmaps configuration, both of which previously had to be configured to set up an ASP.NET handler mapping.
3. **The application Web.config file defines <identity impersonate="true" /> configuration.**  
   The application impersonates client credentials, which is most common with intranet applications. In Integrated mode, client impersonation is not available in some early request-processing stages. In the majority of cases, this is not a problem and you can turn off the migration error. Otherwise, you must configure this application to run by using the Classic ASP.NET mode.

When the migration error is generated, it can generally either migrate the application configuration (recommended in cases 1 and 2, above), or move the application to use Classic ASP.NET mode (in case 3).

Migrating the Application Configuration

IIS 7 migrates the application by using the AppCmd.exe command line tool to perform the migration. The migration error message contains the command that is executed in the command line window, which you must run with administrator user rights, to instantly migrate your application to Integrated mode.

The basic format of the migration command is as follows:

%windir%\system32\inetsrv\APPCMD.EXE migrate config <Application Path>

Where <Application Path> is the virtual path of the application that contains the site name, such as "Default Web Site/app1".

When the migration is complete, your application will run in both Integrated and Classic modes without any problems.

**Note:** If you change the configuration after migration, the server will not prompt you to migrate again. After the initial migration, you must make sure that your configuration remains in sync between the two modes. You can manually migrate the application again by using the AppCmd.exe command line tool.

Moving Back to Classic ASP.NET Integration Mode

If you would rather return to the Classic ASP.NET mode, move your application to an application pool that is configured to run in Classic mode. Other applications continue to run in the new Integrated mode side-by-side with the Classic mode application.

For more information about how to move an application to the Classic ASP.NET mode, see Changing ASP.NET Modes for an Application.

Disabling the Migration Error Message

If you have manually migrated your configuration or want to remain in Integrated mode without migrating your configuration (not recommended), you can disable the migration error message by adding the following to the application's Web.config file:

<system.webServer>   
<validation validateIntegratedModeConfiguration="false" />   
</system.webServer>

**Note**: The server automatically disables the error message after migrating the configuration with the AppCmd.exe command line tool.

If you manually disable the migration error message, you must make sure that your application works properly in Integrated mode, because the server will no longer provide any warnings about the unsupported configuration.

Changing ASP.NET Modes for an Application

If your application does not work properly in Integrated ASP.NET mode, you can move it to the Classic ASP.NET mode by moving it to another application pool. Each application pool is individually configured to use the desired ASP.NET integration mode. This lets you run groups of applications that use different ASP.NET integration modes side-by-side on the same server.

**To change the ASP.NET integration mode for an application:**

1. **Find or create an application pool that is configured to use the desired mode.**   
   By default, all new application pools run in Integrated mode.  
   The ASP.NET set up provides an application pool named "**Classic .NET AppPool**" that runs in the classic ASP.NET integration mode. You can use this application pool for applications that should not run in Integrated mode.  
   You can also change the ASP.NET mode of the existing application pool by using the IIS 7 Administration Tool or the AppCmd.exe command line tool, or by manually editing the application pool configuration.
2. **Set the application to use that application pool.**   
   Each application is configured to use a particular application pool. By default, all applications use the default application pool named "**DefaultAppPool**", which runs in Integrated mode.  
   You can change the application pool of an application by using the IIS 7 Administration Tool or the AppCmd.exe command line tool, or by manually editing the application configuration.

Selecting an ASP.NET Version for an Application

Historically, IIS has supported multiple versions of ASP.NET / CLR side-by-side. For example, IIS allows the same server to run ASP.NET applications by using .NET Framework v1.1 and v2.0. This support was provided by mapping a corresponding version of ASPNET\_isapi.dll to serve requests for the ASP.NET content by using the IIS 7 scriptmaps configuration.

For example, you can use the following scriptmap configuration to enable side-by-side support:

1. **ASP.NET v1.1 application on /app1:**  
   \*.aspx -> d:\WINDOWS\Microsoft.NET\Framework\v1.1.4322\aspnet\_isapi.dll
2. **ASP.NET v2.0 application on /app2:**  
   \*.aspx -> d:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\aspnet\_isapi.dll

When the application receives a \*.aspx request, IIS 7 loads the specified aspnet\_isapi.dll, which loads the correct CLR version into the worker process, and processes the request.

ASP.NET also provides the following ways to configure side-by-side scriptmaps:

1. **ASP.NET MMC extension.** When you pick the version of ASP.NET to use, the extension automatically configures the scriptmaps for the application to point to the correct version of aspnet\_isapi.dll.
2. **ASP.NET aspnet\_regiis.exe.** Use the –i and –r options to install the scriptmaps that point to the corresponding ASP.NET version.

Unfortunately, due to the limitation of the ability to load only one CLR version into a single worker process, you must make sure that two applications that use different versions of ASP.NET are never configured to exist within the same application pool. When this common mistake is made, the first request loads the CLR of the corresponding aspnet\_isapi.dll, and subsequent requests to the other version within the same application pool will fail.

IIS 7 recognizes that the application pool is the ASP.NET version that you select to run an application under. As such, the version of the CLR / ASP.NET that is loaded in the application pool is explicitly configured in the application pool configuration. By default, IIS 7 pre-loads the CLR that is specified by this setting when loading the worker process (unless the version is configured to be empty).

Because application pools are the .NET Framework versioning boundary, you can change the version of an ASP.NET application by doing the following:

1. **Move the application to an application pool that uses the desired ASP.NET version.**   
   By default, applications use the default application pool named "DefaultAppPool," which runs ASP.NET v2.1 in Integrated mode. Move the application to the "Classic .NET AppPool" to run in ASP.NET v2.1 Classic mode, or another application pool of your choice.
2. **Configure the application pool within which the application is running to use the desired ASP.NET version.**   
   By default, all new application pools are configured to run ASP.NET v2.1 in Integrated mode.

**Note:** Do not use aspnet\_regiis /i or /r options to configure the version of ASP.NET for a particular application, or globally.

IIS 7 uses pre-conditioned handler mappings to automatically select the correct set of handler mappings (to ASPNET\_isapi.dll in Classic mode or directly to managed handler types in Integrated mode) depending on the configured CLR version and managed integration mode of the application pool. The ASP.NET 2.0 setup installs these handler mappings at the server level.

Taking Advantage of Integrated ASP.NET Mode

On IIS 7, ASP.NET applications run in Integrated mode by default. However, to take advantage of the benefits that are provided by tighter integration, you must make some modifications to the application configuration. You can also develop new ASP.NET components that take advantage of the Integrated mode to provide powerful functionality to your application.

Enabling ASP.NET Services for All Types of Content

In Integrated mode, services that are provided by ASP.NET modules are available to requests for all content types. However, by default, ASP.NET modules are configured to execute only for requests to ASP.NET content for backward compatibility.

To do this, attach a managedHandler precondition to each ASP.NET module in the configuration section at the server level:

<modules>   
<add name="FormsAuthentication" type="System.Web.Security.FormsAuthenticationModule"   
preCondition="managedHandler" />  
<add name="UrlAuthorization" type="System.Web.Security.UrlAuthorizationModule"   
preCondition="managedHandler" />  
...  
</modules>

The precondition is a rule that the server evaluates on every request to determine whether a module will be executed. The managedHandler precondition is only true for requests to content types that are mapped to ASP.NET handlers.

To apply functionality that is provided by an ASP.NET module to all requests, remove the module entry, and then re-add it without the precondition in the application's root Web.config file.

To enable ASP.NET Forms authentication for the entire application, modify your Web.config file as follows:

<modules>   
<remove name="FormsAuthentication" />   
<add name="FormsAuthentication" type="System.Web.Security.FormsAuthenticationModule" />   
…  
</modules>

With this change, the FormsAuthentication module executes for all requests in your application. This lets you protect all of the application content with Forms authentication. For more information about how to leverage the Integrated mode to provide Forms authentication for all application content, see [Taking Advantage of Integrated Pipeline Mode walkthrough](http://learn.iis.net/page.aspx/244/how-to-take-advantage-of-the-iis7-integrated-pipeline/).

You can also use a shortcut to enable all managed (ASP.NET) modules to run for all requests in your application, regardless of the "managedHandler" precondition. To enable all managed modules to run for all requests without configuring each module entry to remove the "managedHandler" precondition, use the **runAllManagedModulesForAllRequests** property in the <modules> section:

<modules runAllManagedModulesForAllRequests="true" />

When you use this, the "managedHandler" precondition has no effect and all managed modules will run for all requests.

Experiment with enabling other ASP.NET modules to apply to your entire application. For example, use the ASP.NET Output Cache module to output cache ASP pages, or use URL Authorization and Role Manager to configure access control rules for your photos. Or, develop your own module to bring the power of ASP.NET to your entire Web site.

Building More Powerful ASP.NET Services

In Integrated mode, ASP.NET modules apply their services to all content. In addition, because ASP.NET modules execute in the unified request-processing pipeline in Integrated mode, they subscribe to the same request-processing stages and perform the same tasks as native server modules. At the same time, ASP.NET APIs remain largely the same, with a few key additions that unlock previously unavailable functionality.

Runtime Fidelity

In Integrated mode, the ASP.NET request-processing stages that are exposed to modules are directly connected to the corresponding stages of the IIS 7 pipeline. The complete pipeline contains the following stages, which are exposed as HttpApplication events in ASP.NET:

1. **BeginRequest**. The request processing starts.
2. **AuthenticateRequest**. The request is authenticated. IIS 7 and ASP.NET authentication modules subscribe to this stage to perform authentication.
3. **PostAuthenticateRequest**.
4. **AuthorizeRequest**. The request is authorized. IIS 7 and ASP.NET authorization modules check whether the authenticated user has access to the resource requested.
5. **PostAuthorizeRequest**.
6. **ResolveRequestCache**. Cache modules check whether the response to this request exists in the cache, and return it instead of proceeding with the rest of the execution path. Both ASP.NET Output Cache and IIS 7 Output Cache features execute.
7. **PostResolveRequestCache**.
8. **MapRequestHandler**. This stage is internal in ASP.NET and is used to determine the request handler.
9. **PostMapRequestHandler**.
10. **AcquireRequestState**. The state necessary for the request execution is retrieved. ASP.NET Session State and Profile modules obtain their data.
11. **PostAcquireRequestState**.
12. **PreExecuteRequestHandler**. Any tasks before the execution of the handler are performed.
13. **ExecuteRequestHandler**. The request handler executes. ASPX pages, ASP pages, CGI programs, and static files are served.
14. PostExecuteRequestHandler
15. **ReleaseRequestState**. The request state changes are saved, and the state is cleaned up here. ASP.NET Session State and Profile modules use this stage for cleanup.
16. **PostReleaseRequestState**.
17. **UpdateRequestCache**. The response is stored in the cache for future use. The ASP.NET Output Cache and IIS 7 Output Cache modules execute to save the response to their caches.
18. **PostUpdateRequestCache**.
19. **LogRequest**. This stage logs the results of the request, and is guaranteed to execute even if errors occur.
20. **PostLogRequest**.
21. **EndRequest**. This stage performs any final request cleanup, and is guaranteed to execute even if errors occur.

By using the familiar ASP.NET APIs, the ability to execute in the same stages as IIS 7 modules makes tasks that were only previously accessible in native ISAPI filters and extensions now possible in managed code.

For example, you can now write modules that do the following:

1. Intercept the request before any processing has taken place, for example rewriting URLs or performing security filtering.
2. Replace built-in authentication modes.
3. Modify the incoming request contents, such as request headers.
4. Filter outgoing responses for all content types.

See [Developing an IIS 7 Module with .NET](http://learn.iis.net/page.aspx/366/developing-iis7-modules-and-handlers-with-the-net-framework/) for a good example of how to extend IIS 7 with a custom ASP.NET authentication module.

Expanded ASP.NET APIs

The ASP.NET APIs remain backward compatible with previous releases, which allows existing modules to work in IIS 7 without modification, and to apply to all application content without code changes.

However, modules that are written for IIS 7 Integrated mode can take advantage of a few additional APIs to enable key request-processing scenarios that were not previously available.

The new **HttpResponse.Headers** collection allows modules to inspect and manipulate response headers that other application components generate. For example, you can change the Content-Type header that is generated by an ASP page to prompt a download dialog box in the browser. The Cookies collection on the HttpResponse class is also enhanced to allow modification of cookies that are generated as part of the request processing, even if they were created outside of ASP.NET.

The **HttpRequest.Headers** collection is write-enabled, which allows modules to manipulate the incoming request headers. Use this to change the behavior of other server components – for example, you can force a localized application to respond to the client in a different language by dynamically changing the value of the Accept-Language header.

Finally, the **HttpRequest.ServerVariables** collection is now writeable, which allows IIS 7 server variables to be set. ASP.NET modules use this functionality to change values that are provided by IIS 7, and to create new server variables that are visible to other application frameworks, like PHP.

Runtime Integration

In addition to the new APIs, Integrated mode enables existing ASP.NET functionality to provide more value to your application.

The tracing facilities that are provided by ASP.NET, including the System.Diagnostics.Trace class and the ASP.NET page tracing feature, now emit trace information into the IIS 7 trace system. This enables the trace information that is generated during request processing by both IIS 7 and ASP.NET components to be placed in a single log file, facilitating better diagnostics of server issues.

The ASP.NET response-filtering feature, which allows responses to be changed by using a Stream interface before it goes to the client, is enhanced to allow filtering of non-ASP.NET content. Therefore, you can use ASP.NET filtering on all server content, or selectively install the filter only for requests to content that you want to process. With this capability, you can write filtering features that inject or censor certain content in the site's responses, regardless of whether this content comes from an ASPX page or a static file.

Summary

ASP.NET Integration in IIS 7 unlocks the full potential of ASP.NET, enabling developers to create powerful server components with the ease and richness of ASP.NET and the .NET Framework. To learn more about how to leverage ASP.NET Integrated mode for existing applications, see [Taking Advantage of the Integrated Pipeline Mode](http://learn.iis.net/page.aspx/244/how-to-take-advantage-of-the-iis7-integrated-pipeline/). For information about how to build new ASP.NET components to extend the server, see [Developing an IIS 7 Module with .NET](http://learn.iis.net/page.aspx/366/developing-iis7-modules-and-handlers-with-the-net-framework/).