Monitoring IIS and ASP.NET based Applications

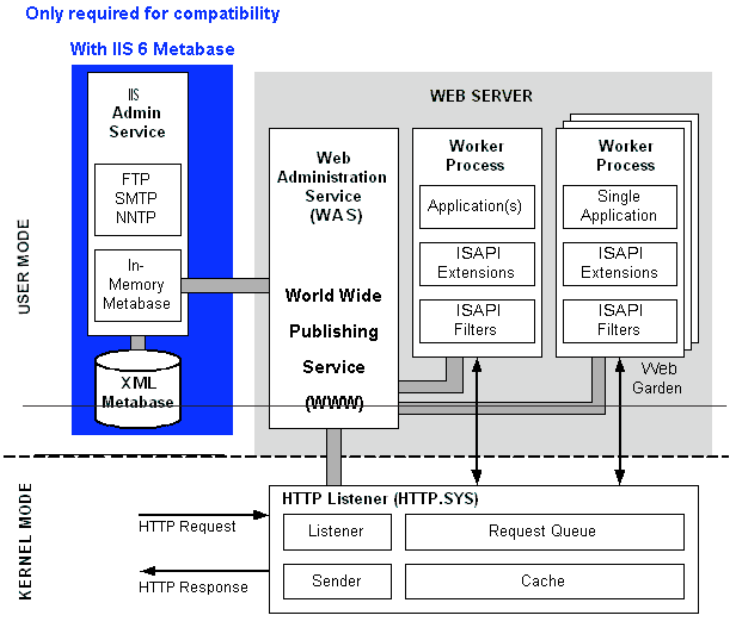
ARGENT

Contents

ASP.NET and IIS Monitoring Overview

Introduction

This section provides an overview of how IIS works and the concepts of how data is moved to and from the client.



1. When a client browser initiates an HTTP request for a resource on the Web server, HTTP.sys intercepts the request.
2. HTTP.sys - contacts WAS to obtain information from the configuration store.
3. WAS requests configuration information from the configuration store, applicationHost.config.
4. WWW Service receives configuration information, such as application pool and site configuration.
5. WWW Service uses the configuration information to configure HTTP.sys.
6. WAS start's a worker process for the application pool to which the request was made?
7. The worker process processes the request and returns a response to HTTP.sys.
8. The client receives a response.

The diagram above illustrates the principal components taking part in the request processing when using the IIS (6 and 7) model.

IIS 6.0 and 7.0 receives HTTP requests in kernel mode and delivers them to the application's isolated worker process.

When an HTTP request arrives at the kernel-mode HTTP Listener (HTTP.sys, bottom), it checks the validity of the request. If the request is invalid, the appropriate HTTP error is immediately returned to the requester. If the request is valid, HTTP.sys checks to see if it can handle the request from its cache. If the response is in the cache, HTTP.sys sends the response immediately. Otherwise, HTTP.sys puts the request in a separate request queue for each worker process (application).

This has many advantages concerning reliability, too. Since running in kernel mode, the request dispatching isn't influenced by crashes and malfunctions happing at user level, that is, in the worker processes. Thereby, even if a worker process crashes, the system is still capable of accepting incoming requests and eventually restarts the crashed process.

It's the worker process who is in charge of loading the ASP.NET ISAPI extension, which, in turn, loads the CRL and delegates all the work to the HTTP Runtime.

The w3wp.exe worker process, differently from the aspnet_wp.exe process used in IIS 5 model, isn't ASP.NET specific, and is used to handle any kind of requests. The specific worker process then decides which ISAPI modules to load according to the type of resources it needs to serve.

Incoming requests are forwarded from the application pool queue to the right worker process via a module loaded in IIS called Web Administration Service (WAS). This module is responsible for reading worker process, web application bindings from the IIS metabase (IIS 6.0 compatibility) and forwarding the request to the right worker process.

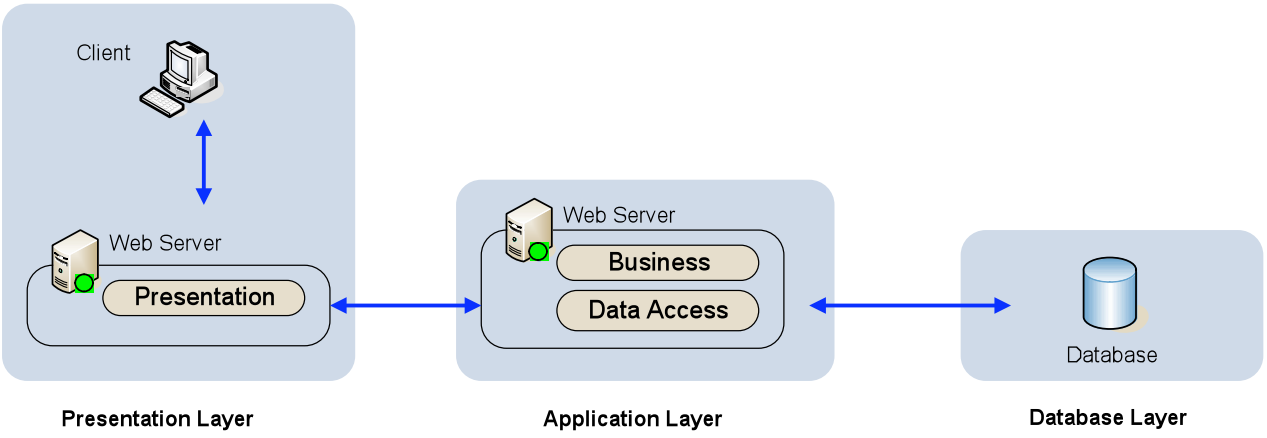
If the necessary worker process is not listening on the request queue, then HTTP.sys signals the Web Administration Service (WAS) to start and configure a worker process based on the configuration information stored in XML format in the metabase(IIS 6.0 compatibility) or applicationhost.config (IIS 7.0).

If a worker process or a group of worker processes (called a Web garden) is already started and connected to the request queue, the worker process pulls the request from the queue, processes it through any ISAPI filter or extension and Web application code, and returns the response to the HTTP.sys and the requester.

The WAS also monitors the health of a worker process and if, for example, the process does not respond or has exceeded a threshold (e.g., the number of hours running or number of requests handled), WAS coordinates with the HTTP Listener to hold requests in the queue while WAS stops the worker process and restarts it.

Typical Web Application Architecture

Typical Web Applications utilise a three-tier architecture which has the following three tiers:



Presentation Layer

This is the topmost level of the application. The presentation tier displays information related to such services as browsing merchandise, purchasing, and shopping cart contents. It communicates with other tiers by outputting results to the browser/client tier and all other tiers in the network. It consists of standard ASP documents, Windows forms, etc. This is the layer that provides an interface for the end user into your application. That is, it works with the results/output of the Business Tier to handle the transformation into something usable and readable by the end user.

Application Layer (Business Logic/Data Access Logic)

The Application logic tier is pulled out from the presentation tier and, as its own layer; it controls an application's functionality by performing detailed processing.

Business Tier - This is basically where the brains of your application reside; it contains things like the business rules, data manipulation, etc. This layer does NOT know anything about HTML, nor does it output it. It does NOT care about ADO or SQL, and it shouldn't have any code to access the database or the like. Those tasks are assigned to each corresponding layer above or below it.

Data Access Tier - This layer is where you will write some generic methods to interface with your data. For example, we will write a method for creating and opening a Connection object (internal). This Layer, obviously, contains no data business rules or data manipulation/ transformation logic. It is merely a reusable interface to the database.

Data Layer

This tier consists of Database Servers. Here information is stored and retrieved. This tier keeps data neutral and independent from application servers or business logic. Giving data its own tier also improves scalability and performance.

Basically, it is the Database Management System (DBMS) – SQL Server, Access, Oracle, MySQL, and plain text (or binary) files, whatever you like. This tier can be as complex and comprehensive as high-end products such as SQL Server and Oracle, which do include the things like query optimization, indexing, etc., all the way down to the simplistic plain text files (and the engine to read and search these files). Some more well-known formats of structured, plain text files include CSV, XML, etc. Notice how this layer is only intended to deal with the storage and retrieval of information. It doesn't care about how you plan on manipulating or delivering this data. This also should include your stored procedures.

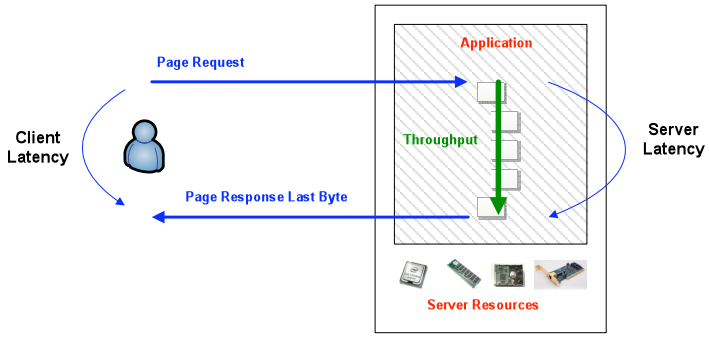
Key Monitoring Concepts

Response Time or Latency

Response time is the amount of time taken to respond to a request. You can measure response time at the server or client as follows:

- **Latency measured at the server.** This is the time taken by the server to complete the execution of a request. This does not take into account the client-to-server latency. This can be measured with the Argent Guardian using some of the Windows Performance counters.
- **Latency measured at the client**. The latency measured at the client includes the request queue, the time taken by the server to complete the execution of the request, and the network latency. You can measure this latency by recording the time taken between requesting a page and loading the last byte and can be measures with the Argent Sentry using the RSP_TREND_ANALYSIS rule.

By measuring latency, you can gauge whether your application takes too long to respond to client requests based on a predefined baseline of how the application best performs.



Throughput

Throughput is the number of requests that can be successfully served by your application per unit time. It can vary depending on the load (number of users) applied to the server. Throughput is usually measured in terms of requests per second. In some systems, throughput may go down when there are many concurrent users. In other systems, throughput remains constant under pressure but latency begins to suffer, perhaps due to queuing. Other systems have some balance between maximum throughput and overall latency under stress.

Resource Utilization

You identify resource utilization costs in terms of server and network resources. The primary resources are the following:

- CPU
- Memory
- Disk I/O
- Network I/O

These can be monitored using the Argent Guardian and collecting windows performance counter data. You can identify the resource cost on a per-operation basis. Operations might include browsing a product catalog, adding items to a shopping cart, or placing an order. You can measure resource costs for a given user load or you can average resource costs when the application is tested using a given *workload profile*.

Workload Profile

A workload profile consists of an aggregate mix of users performing various operations. For example, for a load of 200 concurrent users, the profile might indicate that 20 percent of users perform order placement, 30 percent add items to a shopping cart, while 50 percent browse the product catalog. This helps you identify and optimize areas that consume an unusually large proportion of server resources.

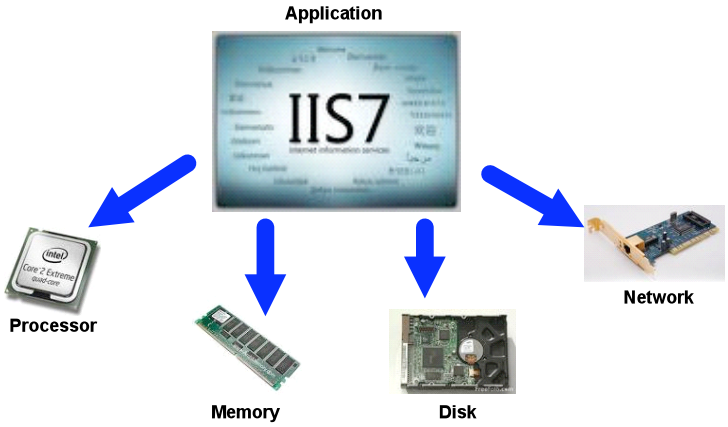
Metrics

Metrics provide information about how close your application is to your performance goals. In addition, they also help you identify problem areas and bottlenecks within your application. You can group various metric types under the following categories:

- **System** - these metrics are related to processor, memory, disk I/O, and network I/O.
- **Platform** - these metrics are related to ASP.NET, and the .NET common language runtime (CLR).
- **Application** - these metrics include custom performance counters – provided as part of the development of the application.
- **Service level** - these are related to your application, such as orders per second and searches per second.

IIS and System Resource Monitoring

When you need to measure how many system resources your application consumes, you need to pay particular attention to the following components that provide 'horsepower' to ensure the application can perform at its best. The following Performance metrics can be collected for reporting and alerting using the Argent Guardian.



Processor

To measure processor utilization you can use the following Windows Performance counters:

Processor\% Processor Time

This counter is the primary indicator of processor activity. High values many not necessarily be bad. However, if the other processor-related counters are increasing linearly such as **Processor Queue Length**, high CPU utilization may be worth investigating. Preferably combine with **System\Processor Queue Length** counter.

Threshold: General figure for the threshold limit 90%

System\Processor Queue Length

If there are more tasks ready to run than there are processors, threads queue up. The processor queue is the collection of threads that are ready but not able to be executed by the processor because another active thread is currently executing. A sustained or recurring queue of more than two threads is a clear indication of a processor bottleneck.

Threshold: Average value consistently higher than 2 usually indicates a bottleneck.

You should use this counter in conjunction with the **Processor\% Processor Time** counter to determine if your application can benefit from more CPUs. There is a single queue for processor time, even on multiprocessor computers.

If the CPU is very busy (90 percent and higher utilization) and the PQL average is consistently higher than 2 per processor as defined below in the Argent Guardian Rule, you may have a processor bottleneck that could benefit from additional CPUs.

Performance Rule Is Broken If
% Processor Time of Processor (_Total) GREATER THAN 90.00 Processor Queue Length of System GREATER THAN 2.00

Or, you could reduce the number of threads and queue more at the application level. This will cause less context switching, and less context switching is good for reducing CPU load. The common reason for a PQL of 2 or higher with low CPU utilization is that requests for processor time arrive randomly and threads demand irregular amounts of time from the processor. This means that the processor is not a bottleneck but that the threading logic that needs to be improved.

Processor\% Privileged Time

This counter indicates the percentage of time a thread runs in privileged mode. When your application calls operating system functions (for example to perform file or network I/O or to allocate memory), these operating system functions are executed in privileged mode.

Performance Rule Is Broken If
% Privileged Time of Processor (_Total) GREATER THAN 75.00

Threshold: A figure that is consistently over 75 percent indicates a bottleneck.

Processor\% Interrupt Time

This counter indicates the percentage of time the processor spends receiving and servicing hardware interrupts. This value is an indirect indicator of the activity of devices that generate interrupts, such as network adapters. A dramatic increase in this counter indicates potential hardware problems.

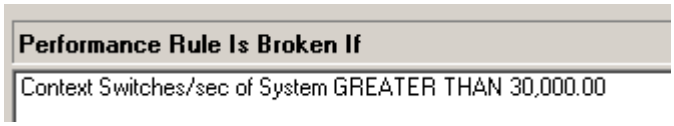
Performance Rule Is Broken If
% Interrupt Time of Processor (_Total) GREATER THAN 50.00

Threshold: Depends on processor. Use Baseline value from application testing.

System\Context Switches/sec

Context switching happens when a higher priority thread preempts a lower priority thread that is currently running or when a high priority thread blocks. High levels of context switching can occur when many threads share the same priority level. This often indicates that there are too many threads competing for the processors on the system. If you do not see much processor utilization and you see very low levels of context switching, it could indicate that threads are blocked.

Below shows an Argent Guardian rule for Context Switches on a dual processor machine.



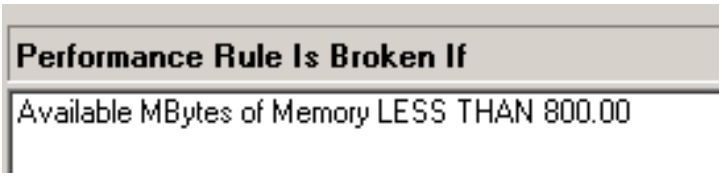
Threshold: As a general rule, context switching rates of less than 5,000 per second per processor are not worth worrying about. If context switching rates exceed 15,000 per second per processor, then there is a constraint.

Memory

To measure memory utilization and the impact of paging, you can use the following performance counters:

Memory\Available Mbytes

This indicates the amount of physical memory available to processes running on the computer. Note that this counter displays the last observed value only. It is not an average.



Threshold: A consistent value of less than 20-25% of installed RAM is an indication of insufficient memory. Need to calculate for each particular machine as there total RAM may vary.

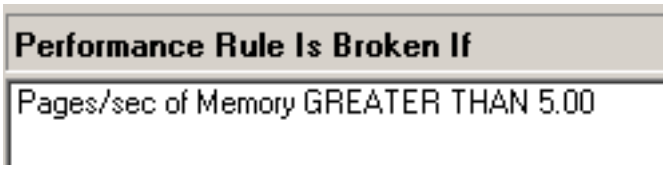
Memory\Page Reads/sec

This counter indicates that the working set of your process is too large for the physical memory and that it is paging to disk. It shows the number of read operations, without regard to the number of pages retrieved in each operation. Higher values indicate a memory bottleneck. If a low rate of page-read operations coincides with high values for **Physical Disk\% Disk Time** and **Physical Disk\Avg. Disk Queue Length**, there could be a disk bottleneck. If an increase in queue length is not accompanied by a decrease in the pages-read rate, a memory shortage exists.

Threshold: Sustained values of five or more indicate a large number of page faults for read requests.

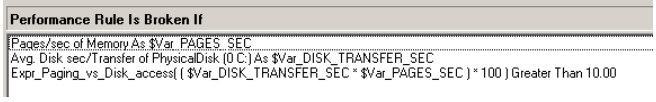
Memory\Pages/sec

This counter indicates the rate at which pages are read from or written to disk to resolve hard page faults. Multiply the values of the **Physical Disk\Avg. Disk sec/Transfer** and **Memory\Pages/sec** counters. If the product of these counters exceeds 0.1, paging is taking more than 10 percent of disk access time, which indicates that you need more RAM.



Threshold: Sustained values higher than five indicate a bottleneck.

Multiply the values of the **Physical Disk\Avg Disk sec/Transfer** and **Memory\Pages/sec** counters. If the product of these counters exceeds 0.1, paging is taking more than 10 percent of disk access time, which indicates that you need more RAM. Below shows the construction of an Argent Guardian Rule to provide tracking of the combined metrics to alert when paging vs. Disk Access is greater that 10%



Memory\Pool Nonpaged Bytes
If there is an increase of 10 percent or more from its value at startup, a serious leak is potentially developing.

Threshold: Watch for an increase of 10% or more from its value at system startup.

Server\Pool Nonpaged Failures
This counter indicates the number of times allocations from the nonpaged pool have failed. It indicates that the computer's physical memory is too small. The nonpaged pool contains pages from a process's virtual address space that are not to be swapped out to the page file on disk, such as a process' kernel object table. The availability of the nonpaged pool determines how many processes, threads, and other such objects can be created. When allocations from the nonpaged pool fail, this can be due to a memory leak in a process, particularly if processor usage has not increased accordingly.

Performance Rule Is Broken If
Pool Nonpaged Failures of Server GREATER THAN 0.000000

Threshold: Regular nonzero values indicate a bottleneck.

Cache\MDL Read Hits %
This counter provides the percentage of Memory Descriptor List (MDL) Read requests to the file system cache, where the cache returns the object directly rather than requiring a read from the hard disk.

Threshold: The higher this value, the better the performance of the file system cache. Values should preferably be as close to 100 percent as possible.

The following counters have no specific threshold and are provided to identify memory issues and values will depend on the system configuration.

Server\Pool Paged Failures
This counter indicates the number of times allocations from the paged pool have failed. This counter indicates that the computer's physical memory or page file is too small.

Memory\Cache Bytes
Monitor the size of cache under different load conditions. This counter displays the size of the static files cache. By default, this counter uses approximately 50 percent of available memory, but decreases if the available memory shrinks, which affects system performance.

Memory\Cache Faults/sec
This counter indicates how often the operating system looks for data in the file system cache but fails to find it. This value should be as low as possible. The cache is independent of data location but is heavily dependent on data density within the set of pages. A high rate of cache faults can indicate insufficient memory or could also denote poorly localized data.

Disk I/O

To measure disk I/O activity, you can use the following counters:

PhysicalDisk\Avg Disk Queue Length
This counter indicates the average number of both read and writes requests that were queued for the selected disk during the sample interval. The following rule shows the setup to alert when Queue length is greater than 4 for a disk volume that uses a RAID 1 array (2 Disk Spindles).

Performance Rule Is Broken If
Avg. Disk Queue Length of PhysicalDisk (0 C:) GREATER THAN 4.00

Threshold: Should not be higher than the number of spindles plus two.

Use the values of the **Avg. Disk Queue Length** and **% Disk Time** counters to detect bottleneck within the disk subsystem. Ensure Avg. Disk Queue Length threshold is set for the number of spindles + 2

Performance Rule Is Broken If
% Disk Time of PhysicalDisk (0 C:) GREATER THAN 90.00 Avg. Disk Queue Length of PhysicalDisk (0 C:) GREATER THAN 4.00

If these two counter values are consistently high, consider doing one of the following:

- Using a faster disk drive
- Moving some files to an additional disk or server.
- Adding disks to a RAID array, if one is being used.

PhysicalDisk\Avg. Disk sec/Transfer
This counter indicates the time, in seconds, of the average disk transfer. This may indicate a large amount of disk fragmentation, slow disks, or disk failures.

Performance Rule Is Broken If
Avg. Disk sec/Transfer of PhysicalDisk (0 C:) GREATER THAN 0.018000

Threshold: Should not be more than 18 milliseconds.

PhysicalDisk\Avg Disk Read Queue Length
This counter indicates the average number of read requests that were queued for the selected disk during the sample interval.

Threshold: Should be less than two.

PhysicalDisk\Avg Disk Write Queue Length
This counter indicates the average number of write requests that were queued for the selected disk during the sample interval.

Performance Rule Is Broken If
Avg. Disk Write Queue Length of PhysicalDisk (0 C:) GREATER THAN 2.00

Threshold: Should be less than two.

PhysicalDisk\Avg Disk sec/Read
This counter indicates the average time, in seconds, of a read of data from the disk.

Threshold: No specific value

PhysicalDisk\Disk Writes/sec
This counter indicates the rate of write operations on the disk.

Threshold: Depends on manufacturer's specification.

Network I/O

To measure network I/O, you can use the following counters:

Network Interface\Bytes Total/sec
This counter indicates the rate at which bytes are sent and received over each network adapter. This counter helps you know whether the traffic at your network adapter is saturated and if you need to add another network adapter. How quickly you can identify a problem depends on the type of network you have as well as whether you share bandwidth with other applications.

Threshold: Sustained values of more than 80 percent of network bandwidth.

Network Interface\Bytes Received/sec and Network Interface\Bytes Sent/sec
This counter indicates the rate at which bytes are received or sent over each network adapter. You can calculate the rate of incoming/outgoing data as a part of total bandwidth. This will help you know that you need to optimize on the incoming/outgoing data from/to the client or that you need to add another network adapter to handle the traffic.

Threshold: No specific value.

Server\Bytes Total/sec
This counter indicates the number of bytes sent and received over the network. Higher values indicate network bandwidth as the bottleneck. If the sum of **Bytes Total/sec** for all servers is roughly equal to the maximum transfer rates of your network, you may need to segment the network.

Threshold: Value should not be more than 50 percent of network capacity.

HTTP.sys

Windows Server 2008, HTTP.sys has the following performance metric counters to help you with monitoring, diagnosing, and capacity planning for Web servers: The HTTP Server API component has the following performance counters to help you with monitoring, diagnosing, and capacity planning for Web servers:

HTTP Service Counters:

- Number of URIs in the cache, added since startup, deleted since startup, number of cache flushes
- Cache hits/second and Cache misses/second

HTTP Service URL Groups:

- Data send rate, data receive rate, bytes transferred (sent and received)
- Maximum # connections, connection attempts rate, GET and HEAD rate requests, total # requests

HTTP Service Request Queues:

- Number of requests in queue, age of oldest requests in queue (age of the last request in the queue)
- Rate of request arrivals into queue, rate of rejection, total # of rejected requests, rate of cache hits

Counter	Parent	Instance
HTTP Service		
CurrentUrisCached	---	---
TotalFlushedUris	---	---
TotalUrisCached	---	---
UriCacheFlushes	---	---
UriCacheHits	---	---
UriCacheMisses	---	---

Counter	Parent	Instance
HTTP Service Request Queues		
ArrivalRate	---	DefaultAppPool
CacheHitRate	---	DefaultAppPool
CurrentQueueSize	---	DefaultAppPool
MaxQueueItemAge	---	DefaultAppPool
RejectedRequests	---	DefaultAppPool
RejectionRate	---	DefaultAppPool

Counter	Parent	Instance
HTTP Service Url Groups		
AllRequests	---	fe00000040000001
BytesReceivedRate	---	fe00000040000001
BytesSentRate	---	fe00000040000001
BytesTransferredRate	---	fe00000040000001
ConnectionAttempts	---	fe00000040000001
CurrentConnections	---	fe00000040000001
GetRequests	---	fe00000040000001
HeadRequests	---	fe00000040000001
MaxConnections	---	fe00000040000001

Note: Only one instance of the HTTP Server API counters exists per machine, as these counters represent the component-wide state.

HTTP Service Url Groups performance counters, the instance will match the Url Group ID. The Url Group ID can be viewed by running **netsh http show servicestate**.

HTTP Service Request Queues performance counters, the instance correspond to Request Queue Name (Application Pool names). The Request Queue Name (if one exists) can be shown by the same **netsh http show servicestate**. However, some server applications may have unnamed Request Queues that cannot be matched to a performance counter instance ID.

The following page show an example of **netsh http show servicestate**.

C:\ netsh http show servicestate

Snapshot of HTTP service state (Server Session View):

```
Server session ID: FF00000020000001
    Version: 2.0
    State: Active
    Properties:
        Max bandwidth: 4294967295
        Timeouts:
            Entity body timeout (secs): 120
            Drain entity body timeout (secs): 120
            Request queue timeout (secs): 65535
            Idle connection timeout (secs): 120
            Header wait timeout (secs): 120
            Minimum send rate (bytes/sec): 240
    URL groups:
    URL group ID: FE00000040000001
        State: Active
        Request queue name: DefaultAppPool
        Properties:
            Max bandwidth: inherited
            Max connections: 4294967295
            Timeouts:
                Entity body timeout (secs): 120
                Drain entity body timeout (secs): 120
                Request queue timeout (secs): 65535
                Idle connection timeout (secs): 120
                Header wait timeout (secs): 0
                Minimum send rate (bytes/sec): 0
            Logging information:
                Log directory: C:\inetpub\logs\LogFiles\W3SVC1
                Log format: 0
            Number of registered URLs: 1
            Registered URLs:
                HTTP://*:80/

Request queues:
    Request queue name: Request queue is unnamed.
        Version: 1.0
        State: Active
        Request queue 503 verbosity level: Basic
        Max requests: 1000
        Number of active processes attached: 1
        Process IDs:            472

    Request queue name: DefaultAppPool
        Version: 2.0
        State: Active
        Request queue 503 verbosity level: Limited
        Max requests: 1000
        Number of active processes attached: 0
        Controller process ID: 1708
        Process IDs:
```

CLR and Managed Code

This section describes what you need to measure in relation to the CLR and managed code and how you capture the key metrics. This applies to all managed code, regardless of the type of assembly, for example, ASP.NET application, Web service, serviced component, and data access component.

When measuring the processes running under CLR some of the key points to look for are as follows:

Memory -

Measure managed and unmanaged memory consumption.

Working set -

Measure the overall size of your application's working set.

Exceptions -

Measure the effect of exceptions on performance.

Contention -

Measure the effect of contention on performance.

Threading -

Measure the efficiency of threading operations

Memory

To measure memory consumption, use the following counters:

Process (w3wp)\Private Bytes

The committed memory owned by this process in bytes. Memory leaks are identified by a consistent and prolonged increase in Private Bytes. This is the best performance counter for detecting memory leaks.

Threshold: the minimum of 60% of physical RAM.

Values greater than 60% of total physical RAM begin to have an impact upon performance, especially during application and process restarts. The likelihood of an OutOfMemoryException greatly increases when Private Bytes exceeds 800 MB in a process with a virtual address space limit of 2 GB or 1800MB in a process with a virtual address space limit of 3GB

Performance Rule Is Broken If
Private Bytes of Process (w3wp) GREATER THAN 1,800,000,000.00

There may be multiple worker processes for the Application. Then use a Regular expression to do the match.

.NET CLR Memory\% Time in GC

This counter indicates the percentage of elapsed time spent performing a garbage collection since the last garbage collection cycle. The most common cause of a high value is making too many allocations, which may be the case if you are allocating on a per-request basis for ASP.NET applications. You need to study the allocation profile for your application if this counter shows a higher value.

Performance Rule Is Broken If
% Processor Time of Processor (_Total) GREATER THAN 70.00 % Time in GC of .NET CLR Memory (w3wp) GREATER THAN 5.00

Threshold: This counter should average about 5 percent for most applications when the CPU is 70 percent busy, with occasional peaks. As the CPU load increases, so does the percentage of time spent performing garbage collection. Keep this in mind when you measure the CPU.

Contention

.NET CLR LocksAndThreads\Contention Rate / sec

This counter displays the rate at which the runtime attempts to acquire a managed lock but without a success. You may want to run dedicated tests for a particular piece of code to identify the contention rate for the particular code path.

Performance Rule Is Broken If
Contention Rate / sec of .NET CLR LocksAndThreads (w3wp) GREATER THAN 0.000000

Threshold: No specific value. Sustained nonzero values may be a cause of concern

.NET CLR LocksAndThreads\Current Queue Length

This counter displays the last recorded number of threads currently waiting to acquire a managed lock in an application. You may want to run dedicated tests for a particular piece of code to identify the average queue length for the particular code path. This helps you identify inefficient synchronization mechanisms

Performance Rule Is Broken If
Current Queue Length of .NET CLR LocksAndThreads (Any Instance) GREATER THAN 0.000000

Threshold: No specific value. Use Baseline Application Tests to Define.

Exceptions

.NET CLR Exceptions\# of Exceps Thrown / sec

This counter indicates the total number of exceptions generated per second in managed code. Exceptions are very costly and can severely degrade your application performance. You should investigate your code for application logic that uses exceptions for normal processing behavior.

Response.Redirect, Server.Transfer, and **Response.End** all cause a **ThreadAbortException** in ASP.NET applications.

Performance Rule Is Broken If
Requests/Sec of ASP.NET Applications (__Total__) As $Var_REQUEST_SEC # of Exceps Thrown / sec of .NET CLR Exceptions (w3wp) As $Var_EXECPTIONS_SEC Expr_ASP_Net_Exceptions(($Var_EXECPTIONS_SEC / $Var_REQUEST_SEC) * 100) Greater Than 5.00

Threshold: This counter value should be less than 5 percent of **Request/sec** for the ASP.NET application. If you see more than 1 request in 20 throw an exception, you should pay closer attention to it.

The # of Exceps Thrown counter displays the number of exceptions thrown in an application, because these can have performance implications. However, some code paths rely on exceptions for proper functioning. For example, the Redirect method on the Response object throws the ThreadAbortException exception, which cannot be caught.

Performance Rule Is Broken If
Errors Total of ASP.NET Applications (__Total__) GREATER THAN 0.000000 # of Exceps Thrown of .NET CLR Exceptions (_Global_) GREATER THAN 0.000000

Therefore, it can be useful to track this value along with the Errors Total counter to see if the exception generated an error in the application.

Process (w3wp)\Handle Count

Threshold: 2000. A handle count of 2000 in w3wp is suspicious, and 10,000 are far beyond acceptable limits. Noticeable performance degradation will occur if the total handle count for all processes exceeds approximately 40,000, which is entirely achievable during a denial-of-service attack against IIS

Performance Rule Is Broken If
Handle Count of Process [w3wp] GREATER THAN 10,000.00

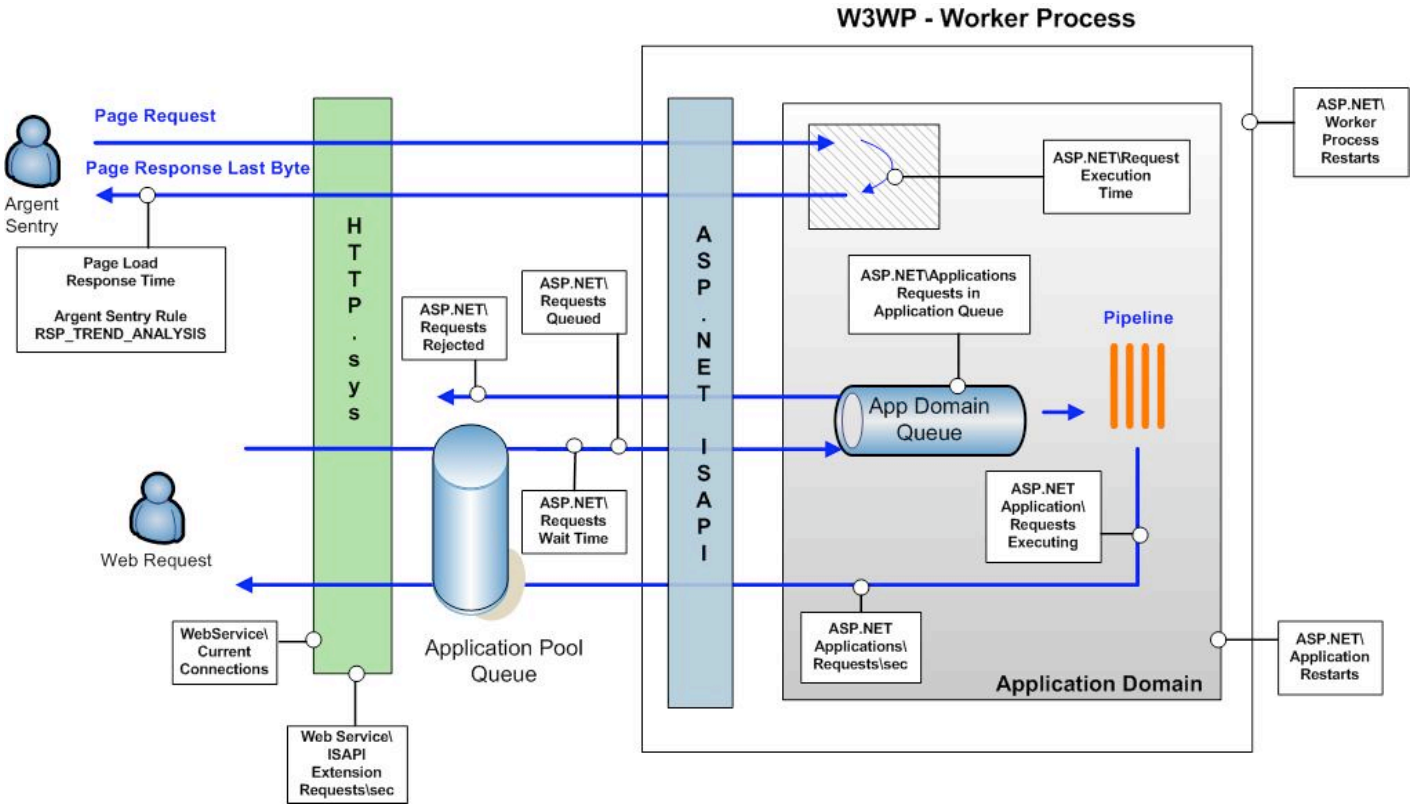
Thread\Thread State

You need to monitor this counter when you fear that a particular thread is consuming most of the processor resources.

Thread State is the current state of the thread. It is 0 for Initialized, 1 for Ready, 2 for Running, 3 for Standby, 4 for Terminated, 5 for Wait, 6 for Transition, 7 for Unknown. A Running thread is using a processor; a Standby thread is about to use one. A Ready thread wants to use a processor, but is waiting for a processor because none are free. A thread in Transition is waiting for a resource in order to execute, such as waiting for its execution stack to be paged in from disk. A Waiting thread has no use for the processor because it is waiting for a peripheral operation to complete or a resource to become free.

IIS and ASP.NET Application Monitoring

Below is a review of performance counters that are useful for monitoring ASP.NET application performance. The following section indicates several performance counters that are generally needed and recommended by Microsoft in their Patterns and Practices for .NET application performance. Some counters For example, the session state and transactions performance counters are only necessary when the features are used.



You measure ASP.NET performance primarily by using system performance counters. The diagram above shows the main performance counters that you use to measure ASP.NET performance and how they relate to the ASP.NET request processing cycle.

A few thresholds are recommended based upon best practice with debugging and testing ASP.NET applications. Web Administrators should determine whether to raise alerts when these thresholds are exceeded based upon their experience. In most cases, alerts are appropriate, especially if the threshold is exceeded for extended periods of time.

At a minimum, the following performance counters should be monitored for Microsoft ASP.NET applications:

- **Web Service\Current Connections**
- **ASP.NET\Request Execution Time**
- **ASP.NET\Request Queued**
- **ASP.NET\Request Wait Time**
- **ASP.NET\Worker Process Restarts**
- **ASP.NET\Application Restarts**
- **ASP.NET\Request Rejected**
- **ASP.NET Applications\Requests in Application Queue**

ASP.NET\Application Restarts

The number of application restarts. Recreating the application domain and recompiling pages takes time, therefore unforeseen application restarts should be investigated. The application domain is unloaded when one of the following occurs:

- Modification of **machine.config**, **web.config**, or **global.asax**.
- Modification of the application's bin directory or its contents.
- When the number of compilations (ASPX, ASCX, or ASAX) exceeds the limit specified by <compilation numRecompilesBeforeAppRestart=/>.
- Modification of the physical path of a virtual directory.
- Modification of the code-access security policy.
- The Web service is restarted.

Performance Rule Is Broken If
Application Restarts of ASP.NET GREATER THAN 0.000000

Threshold: 0 (in a perfect world, the application domain will survive for the life of the process. Excessive values should be investigated.)

ASP.NET\Requests Rejected

The number of requests rejected. Requests are rejected when one of the queue limits is exceeded (see description of Requests Queued). Requests can be rejected for a number of reasons. Backend latency, such as that caused by a slow SQL server, is often preceded by a sudden increase in the number of pipeline instances and a decrease in CPU utilization and Requests/sec. A server may be overwhelmed during times of heavy load due to processor or memory constraints that ultimately result in the rejection of requests

Performance Rule Is Broken If
Requests Rejected of ASP.NET GREATER THAN 0.000000

Threshold: 0. (the value of this counter should be 0. Values greater should be investigated.)

Note: requests are rejected when the Requests Current counter exceeds the Request Queue Limit, when this limit is exceeded, requests will be rejected with a 503 status code and the message "Server is too busy." If a request is rejected for this reason, it will never reach managed code, and error handlers will not be notified. Normally this is only an issue when the server is under a very heavy load, although a "burst" load every hour might also cause this

ASP.NET\Worker Process Restarts

The number of w3wp process restarts.

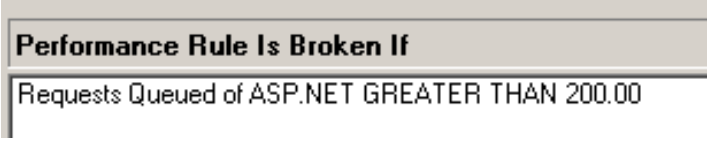
Performance Rule Is Broken If
Worker Process Restarts of ASP.NET (DeltaSinceLastPoll) GREATER THAN 0.000000

Threshold: 1. Process restarts are expensive and undesirable. Values are dependent upon the process model configuration settings, as well as unforeseen access violations, memory leaks, and deadlocks. Requests will be lost if an access violation or deadlock occurs. If process model settings are used to preemptively recycle the process, it will be necessary to set an appropriate threshold.

ASP.NET\Requests Queued.

The number of requests currently queued. When running on IIS 6.0, there is a queue where requests are posted to the managed Thread Pool from native code, and a queue for each virtual directory. This counter includes requests in all queues. On IIS 6.0 it increases when there are incoming requests and a shortage of worker threads.

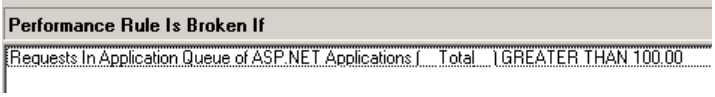
By itself, this counter is not a clear indicator of performance issues, nor can it be used to determine when requests will be rejected. See other counters Requests Current, Requests in Application Queue and Requests Rejected.



Threshold: Depends on business application requirements.

ASP.NET Applications\Requests in Application Queue

The number of requests in the application request queue (see description of Requests Queued above). In addition to Requests Current, Requests in Application Queue provides a warning for when requests will be rejected. If there are only a couple virtual directories, increasing the default **appRequestQueueLimit** to 200 or 300 may be suitable, especially for slow applications under heavy load.

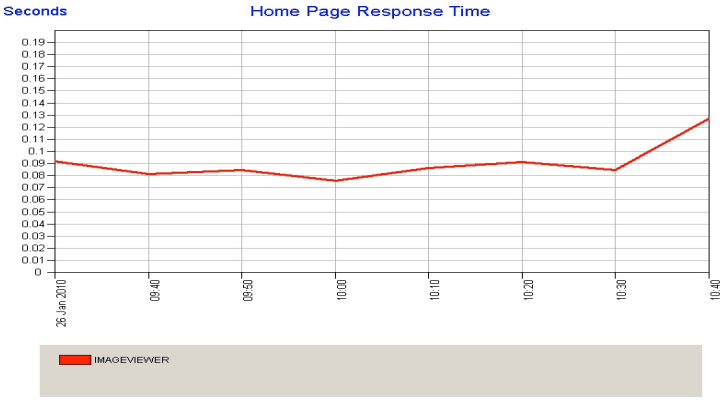


Threshold: Depends on business requirements.

Response Time and Latency

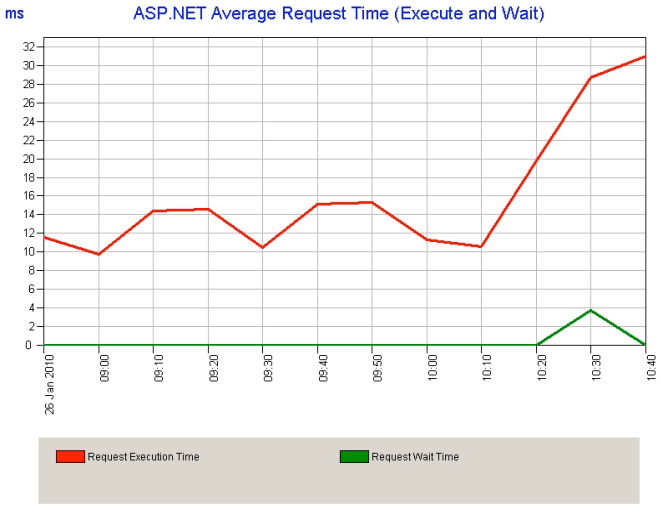
You can measure response time (and latency) from a client and server perspective. From the client perspective, you can measure the time taken for the first byte of the response to reach the client and the time taken for the last time to reach the client.

The latency here includes network latency (the time taken for the request and response to travel over the network) and server latency (the time taken for the server to process the request.) You measure time to first byte (TTFB) and time to last byte (TTLB) by using Argent Sentry to open a web page and record the time taken to load the page – the response time can be used to trigger alerts and also can be stored into the database for trending purposes.



The above chart shows the average page load time for a web site to load the home page and as shown here the average appears to be around 100ms.

On the server-side, you measure the time taken by ASP.NET to process a request by using the **ASP.NET\Request Execution Time** performance counter – this can be tracked using Argent Guardian and a Performance rule against the IIS server.



This data can be used to trigger alerts if the average request execution time exceeds a value that is deemed to be acceptable within the web applications parameters. Data can also be captured using the Argent Predictor as shown above to chart the value over time.

This data then can be correlated with the home page load time to identify when the application is taking longer to load than under normal conditions.

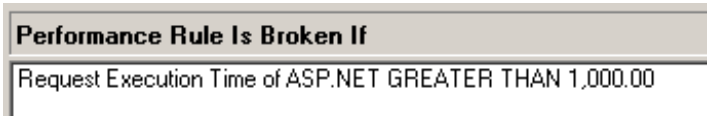
In this chart we have a Request Execution time of around 30ms @ 10:40 this appeared to raise as to did the Request Wait Time counter, if we look at the Page Response time on the previous page it also increased from around 90ms to 130ms @ 10:40.

From these pieces of data we can see how long the client and network processing time has taken by subtracting the Request Execution time from the Home Page Response Time

130ms (Home Page Response Time) - 30ms (Request Execution Time) = 100ms (client and network latency)

ASP.NET\Request Execution Time

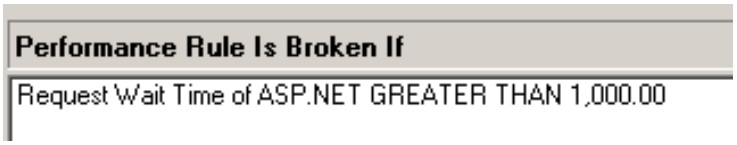
The time taken to execute the last request (milliseconds). ASP.NET\ Request Execution Time is an instance counter, and very volatile.



Threshold: Depends on Business Application Requirements. The value of this counter should be stable. Experience will help set a threshold for a particular site. When the process model is enabled, the request execution time includes the time required to write the response to the client, and therefore depends upon the bandwidth of the client's connection.

ASP.NET\Request Wait Time

The amount of time (milliseconds) that the most recent request spent waiting in the queue, or named pipe, that exists between inetinfo and w3wp (see description of Requests Queued). This does not include any time spent waiting in the application queues.



Threshold: 1000. The average request should spend 0 milliseconds waiting in the queue.

Throughput

To measure ASP.NET application throughput, use the following counters:

ASP.NET Applications\Requests/Sec

The throughput of the ASP.NET application. It is one the primary indicators that help you measure the cost of deploying your system at the necessary capacity.

Threshold: Depends on your business requirements should have a baseline recorded during testing.

Web Service\ISAPI Extension Requests/sec

The rate of ISAPI extension requests that are simultaneously being processed by the Web service. This counter is not affected by the ASP.NET worker process.

Threshold: Depends on your business requirements should have a baseline recorded during testing.

Building a model of a well running application by using a table similar to the following will provide an application workload profile – this can be used to determine what thresholds should be set and at what values the application starts to perform badly.

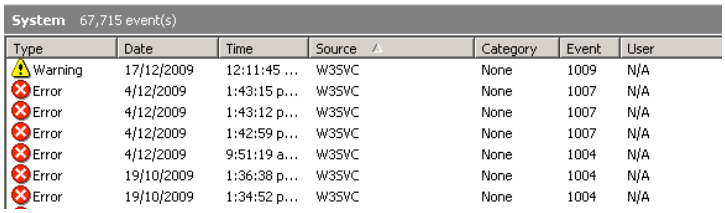
Object	Counter	Test 1	Test 2	Set Threshold
Connections	Web Service\Current Connections	50	200	
CPU	Processor\% Processor Time System\Processor Queue Length System\Context Switches/sec	40 1 2000	65 3 4000	
Disk	Avg. Disk Queue Length %Disk Time	3 35	5 60	
Memory	Memory\Pages/sec Memory\Available Mbytes	2 500	8 300	
Network	Network Interface\Bytes Total/sec	125000	370000	
Application	ASP.NET Applications\Requests/Sec ASP.NET Applications\Requests in App Q ASP.NET\Request Execution Time ASP.NET\Request Wait Time	37 2 370ms 0ms	100 5 650ms 50ms	
Client	Argent Sentry Page Load Time (seconds)	2	6	

Monitoring the Windows Event Logs

It is critical to monitor the event log for messages from ASP.NET and Microsoft Internet Information Server (IIS). ASP.NET writes messages to the application log, for example, IIS writes messages to both the application and/or system logs, for example, each time the w3wp worker process reports itself unhealthy or crashes. It is quite easy to use the Argent Data Consolidator that reads the application log and filters out messages from ASP.NET and IIS, and fires an alert (sends e-mail or SMS) if necessary.

WWW Service Events

These are generated by the part of the World Wide Web Publishing Service (WWW service) that handles internal administration of the W3SVC. The WWW service events are listed in the System log, with the source name W3SVC.

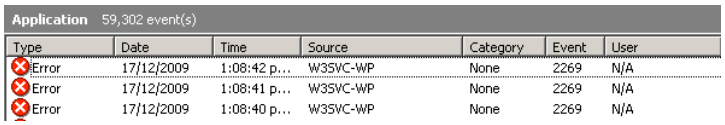


System 67,715 event(s)

Type	Date	Time	Source	Category	Event	User
Warning	17/12/2009	12:11:45 ...	W3SVC	None	1009	N/A
Error	4/12/2009	1:43:15 p...	W3SVC	None	1007	N/A
Error	4/12/2009	1:43:12 p...	W3SVC	None	1007	N/A
Error	4/12/2009	1:42:59 p...	W3SVC	None	1007	N/A
Error	4/12/2009	9:51:19 a...	W3SVC	None	1004	N/A
Error	19/10/2009	1:36:38 p...	W3SVC	None	1004	N/A
Error	19/10/2009	1:34:52 p...	W3SVC	None	1004	N/A

WWW Service Worker Process Events

The WWW service worker process generates, such as authentication and authorization, application problems, memory monitoring, and so on. These events are listed in the Application log with the source name W3SVC-WP.

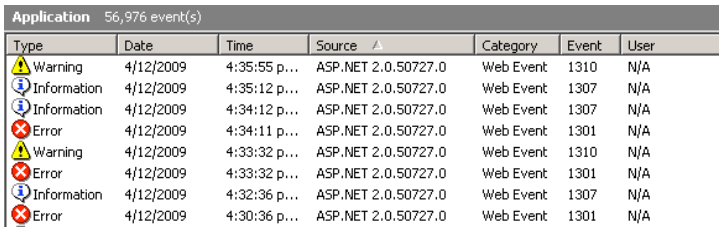


Application 59,302 event(s)

Type	Date	Time	Source	Category	Event	User
Error	17/12/2009	1:08:42 p...	W3SVC-WP	None	2269	N/A
Error	17/12/2009	1:08:41 p...	W3SVC-WP	None	2269	N/A
Error	17/12/2009	1:08:40 p...	W3SVC-WP	None	2269	N/A

ASP.NET Events

The ASP.NET process generates events, such as application problems, thread, and exception monitoring, and so on. These events are listed in the Application log with the source name ASP.NET (Version).

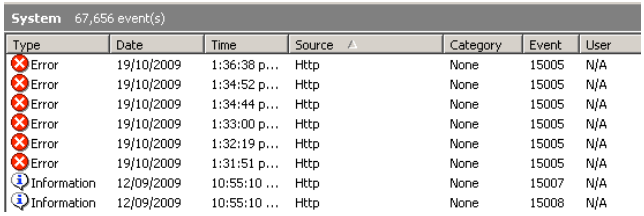


Application 56,976 event(s)

Type	Date	Time	Source	Category	Event	User
Warning	4/12/2009	4:35:55 p...	ASP.NET 2.0.50727.0	Web Event	1310	N/A
Information	4/12/2009	4:35:12 p...	ASP.NET 2.0.50727.0	Web Event	1307	N/A
Information	4/12/2009	4:34:12 p...	ASP.NET 2.0.50727.0	Web Event	1307	N/A
Error	4/12/2009	4:34:11 p...	ASP.NET 2.0.50727.0	Web Event	1301	N/A
Warning	4/12/2009	4:33:32 p...	ASP.NET 2.0.50727.0	Web Event	1310	N/A
Error	4/12/2009	4:33:32 p...	ASP.NET 2.0.50727.0	Web Event	1301	N/A
Information	4/12/2009	4:32:36 p...	ASP.NET 2.0.50727.0	Web Event	1307	N/A
Error	4/12/2009	4:30:36 p...	ASP.NET 2.0.50727.0	Web Event	1301	N/A

HTTP Events

The HTTP process generates events, such as authentication and authorization, application problems, memory monitoring, and so on. These events are listed in the System log with the source name Http.



System 67,656 event(s)

Type	Date	Time	Source	Category	Event	User
Error	19/10/2009	1:36:38 p...	Http	None	15005	N/A
Error	19/10/2009	1:34:52 p...	Http	None	15005	N/A
Error	19/10/2009	1:34:44 p...	Http	None	15005	N/A
Error	19/10/2009	1:33:00 p...	Http	None	15005	N/A
Error	19/10/2009	1:32:19 p...	Http	None	15005	N/A
Error	19/10/2009	1:31:51 p...	Http	None	15005	N/A
Information	12/09/2009	10:55:10 ...	Http	None	15007	N/A
Information	12/09/2009	10:55:10 ...	Http	None	15008	N/A

Adding ASP.NET Startup and Shutdown Application Events

By altering the root web.config you can get an event for every application shutdown and start up. This is a good way to get more detailed information on *why* the application shut down.

Open up the root web.config, (located in the %WinDir%\Microsoft.NET\Framework\v2.0.50727\CONFIG directory,) locate the healthMonitoring.rules subkey and add the following:

```
<add name="Application Lifetime Events Default"
eventName="Application Lifetime Events"
provider="EventLogProvider" profile="Default" minInstances="1"
maxLimit="Infinite" minInterval="00:01:00" custom="" />
```

Now when the application exited for an application-specific reason you'll get an event like this:

```
Event code: 1002
Event message: Application is shutting down. Reason: Configuration changed.
Event time: 2/14/2008 10:00:41 AM
Event time (UTC): 2/14/2008 9:00:41 AM
Event ID: a1314c10a0c84222ae2d870d85308304
Event sequence: 18
Event occurrence: 1
Event detail code: 50004

Application information:
    Application domain: /LM/w3svc/1/ROOT/Test-1-128474532435626182
    Trust level: Full
    Application Virtual Path: /Test
    Application Path: c:\inetpub\wwwroot\Test\
    Machine name: TEST
```

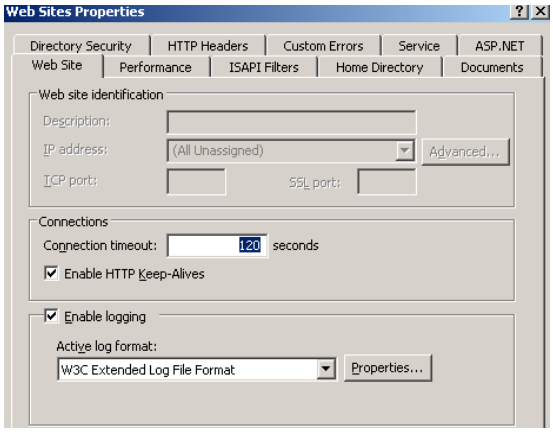
As you can see the application shut down because the configuration changed. Note that you won't get an event if you manually kill the entire application pool in IIS manager, or similar. One thing that you *will* get, however, is the following event each and every time the application starts up again:

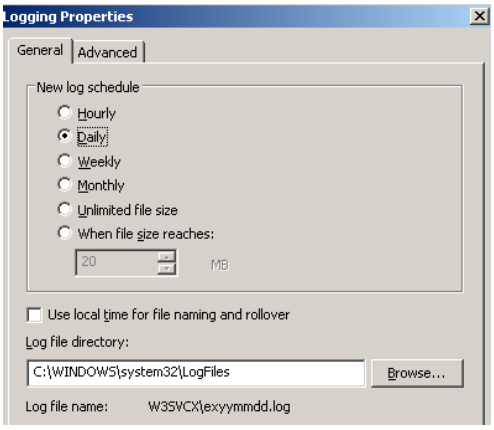
```
Event code: 1001
Event message: Application is starting.
Event time: 2/14/2008 10:00:47 AM
Event time (UTC): 2/14/2008 9:00:47 AM
Event ID: 1f41fd3b17764330ac61804094b0abf0
Event sequence: 1
Event occurrence: 1
Event detail code: 0

Application information:
    Application domain: /LM/w3svc/1/ROOT/Test-1-128474532435626182
    Trust level: Full
    Application Virtual Path: /Test
    Application Path: c:\inetpub\wwwroot\Test\
    Machine name: TEST
```

So even if the application shut down for a reason that didn't generate an event, (IISReset, idle server, etc.) you'll at least see that for some reason it had to start up again.

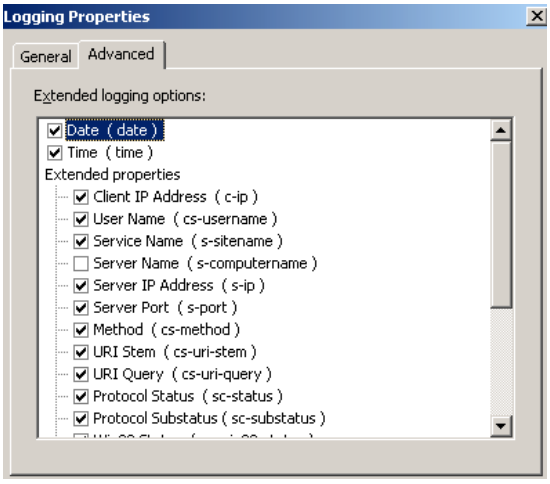
Monitoring the W3C Log

First, enable W3C logging for IIS through the Internet Information Services (IIS) Manager.





This log can be configured to include various data about the requests, such as the URI, status code etc.



Scan the log for error codes such as 404 Not Found, and take action to correct links, if necessary.

On IIS 6.0, the substatus code is included in the log and is useful for debugging. IIS uses substatus codes to indentify specific problems.

For example, 404.2 indicate that the ISAPI extension handling the request is locked down.

Example W3C Log File

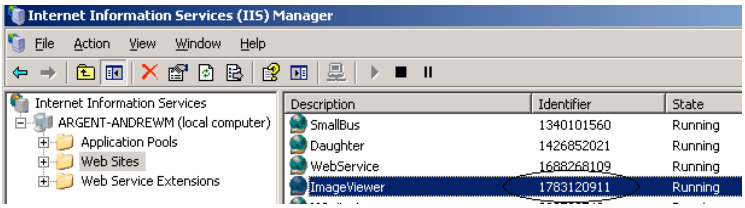
#Fields: date time s-sitename s-ip cs-method cs-uri-stem cs-uri-query s-port cs-username c-ip cs(User-Agent) sc-status sc-substatus sc-win32-status

2010-01-27 00:00:08 W3SVC1783120911 127.0.0.1 GET /HTML/main.htm - 8282 - 127.0.0.1 Sentinel 200 0 0

2010-01-27 00:00:08 W3SVC1783120911 127.0.0.1 GET /HTML/main.htm - 8282 - 127.0.0.1 Sentinel 200 0 0

W3C Logs are located in a directory under the following folder C:\WINDOWS\system32\LogFiles.

However a directory is used for logging each website separately and is named by using the web site identifier.



So the directory with the W3C logs for the ImageViewer website would be.

C:\WINDOWS\system32\LogFiles\W3SVC1783120911

In the **W3C Logging Fields** dialog box, select one or more of the following options:

Date (date):	the date on which the request occurred.
Time (time):	the time, in (UTC), at which the request occurred.
Client IP Address (c-ip):	the IP address of the client that made the request.
User Name (cs-username):	the authenticated user. Anonymous users are indicated by a hyphen.
Service Name (s-sitename):	the site instance number that fulfilled the request.
Server Name (s-computername):	the name of the server on which the log files entry was generated.
Server IP Address (s-ip):	the IP of the server on which the log file entry was generated.
Server Port (s-port):	the server port number that is configured for the service.
Method (cs-method):	the requested action, for example, a GET method.
URI Stem (cs-uri-stem):	the Universal Resource Identifier, or target, of the action.
URI Query (cs-uri-query):	the query, if any, that the client was trying to perform. A Universal Resource Identifier (URI) query is necessary only for dynamic pages.
Protocol Status (sc-status):	the HTTP or FTP status code.

HTTP Status Codes

Status Code	Condition
100	Informational
200	Successful
300	Redirection
400	Client Error
500	Server Error

Protocol Sub-status (sc-substatus): the HTTP or FTP substatus code.

HTTP 400 - Class Client Error Codes Returned by IIS	
Status Code	**Condition**
400	Cannot resolve the request.
401.x	Unauthorized.
403.x	Forbidden.
404.x	File or directory not found.
405	HTTP verb used to access this page is not allowed.
406	Client browser does not accept the MIME type of requested page.
407	Initial proxy authentication required by the Web server.
412	Precondition set by the client failed when evaluated on the Web server.
413	Request entity too large.
414	Request URL is too large and therefore unacceptable on the Web server.
415	Unsupported media type.
416	Requested range not satisfiable
417	Expectation failed.
423	Locked error.

Win32 Status (sc-win32-status):	the Windows status code.
Bytes Sent (sc-bytes):	the number of bytes that the server sent.
Bytes Received (cs-bytes):	the number of bytes that the server received.
Time Taken (time-taken):	the length of time that the action took in milliseconds.
Protocol Version (cs-version):	the protocol version, HTTP or FTP, that the client used.
Host (cs-host):	the host name, if any.
User Agent (cs(UserAgent)):	the browser type that the client used.
Cookie (cs(Cookie)):	the content of the cookie sent or received, if any.
Referer (cs(Referer)):	the site that the user last visited. This site provided a link to the current site.

Monitoring the HTTPERR Log

Malformed or bad requests and requests that fail to be served by an Application Pool are logged to the HTTPERR log by HTTP.SYS, the kernel-mode driver for handling HTTP requests. Each entry includes the URL and a brief description of the error.

W3C Logs are located in a directory under the following folder *C:\WINDOWS\system32\LogFiles\HTTPERR*.

The following sample lines are from an HTTP API error log:

2002-07-05 18:45:09 172.31.77.6 2094 172.31.77.6 80 HTTP/1.1 GET /qos/1kbfile.txt 503 – ConnLimit 2002-07-05 19:51:59 127.0.0.1 2780 127.0.0.1 80 HTTP/1.1 GET /ThisIsMyUrl.htm 400 – Hostname 2002-07-05 19:53:00 127.0.0.1 2894 127.0.0.1 80 HTTP/2.0 GET / 505 - Version_N/S 2002-07-05 20:06:01 172.31.77.6 64388 127.0.0.1 80 - - - - - Timer_MinBytesPerSecond

Types of errors that the HTTP API logs

The HTTP API logs error responses to clients, connection time-outs, orphaned requests, rejected requests and dropped connections that are handled incorrectly.

The following list identifies the types of errors that the HTTP API logs:

- **Responses to clients** The HTTP API sends an error response to a client, for example, a 400 error that is caused by a parse error in the last received request. After the HTTP API sends the error response, it terminates the connection.
- **Connection time-outs** The HTTP API times out a connection. If a request is pending when the connection times out, the request is used to provide more information about the connection in the error log.
- **Orphaned requests** A user-mode process quits unexpectedly while there are still queued requests that are routed to that process. The HTTP API logs the orphaned requests in the error log.
- **Rejected requests**. Requests are rejected by HTTP.SYS when the kernel request queue is exceeded, and when the application is taken offline by the Rapid Fail Protection feature. When the first issue occurs, the URL is logged with the message **QueueFull**, and when the second occurs, the message is **App Offline**. By default, the kernel request queue is set to 1,000, and can be configured on the Application Pool Properties page in IIS Manager. It is recommended to increase this to 5,000 for a busy site, since the kernel request queue could easily surpass 1,000 if an Application Pool crashes while a site is under a very high load.

- **Lost Requests** due to a worker process crash or hang. When this occurs the URL will be logged with the message, **Connection_Abandoned_By_AppPool**, for each in-flight request. An in-flight request is one that was sent to a worker process for processing, but did not complete before the crash or hang.

Format of the HTTP API error logs

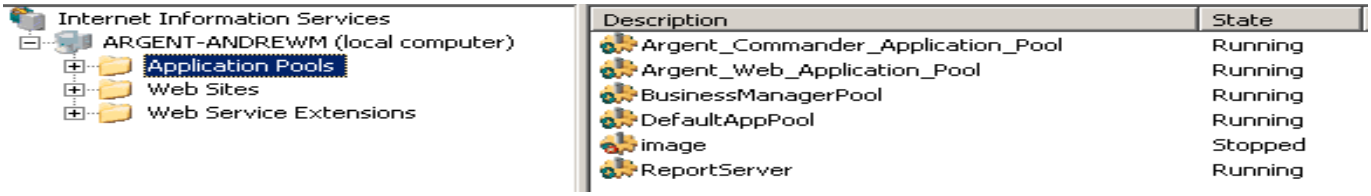
Generally, HTTP API error log files have the same format as W3C error logs, except that HTTP API error log files do not contain column headings. Each line of an HTTP API error log records one error. The fields appear in a specific order. A single space character separates each field from the previous field. The following table identifies the fields and the order of the fields in an error log record.

Field	Description
Date	The Date field follows the W3C format. This field is based on Coordinated Universal Time (UTC). The Date field is always ten characters in the form of YYYY-MM-DD. For example, May 1, 2003 is expressed as 2003-05-01.
Time	The Time field follows the W3C format. This field is based on UTC. The time field is always eight characters in the form of MM:HH:SS. For example, 5:30 PM (UTC) is expressed as 17:30:00.
Client IP Address	The IP address of the affected client. The value in this field can be either an IPv4 address or an IPv6 address. If the client IP address is an IPv6 address, the ScopeId field is also included in the address.
Client Port	The port number for the affected client.
Server IP Address	The IP address of the affected server. The value in this field can be either an IPv4 address or an IPv6 address. If the server IP address is an IPv6 address, the ScopeId field is also included in the address.
Server Port	The port number of the affected server.
Protocol Version	The version of the protocol that is being used. If the connection has not been parsed sufficiently to determine the protocol version, a hyphen (0x002D) is used as a placeholder for the empty field. If either the major version number or the minor version number that is parsed is greater than or equal to 10, the version is logged as HTTP/?.?.
Verb	The verb state that the last request that is parsed passes. Unknown verbs are included, but any verb that is more than 255 bytes is truncated to this length. If a verb is not available, a hyphen (0x002D) is used as a placeholder for the empty field.

Field	Description
CookedURL + Query	The URL and any query that is associated with it are logged as one field that is separated by a question mark (0x3F). This field is truncated at its length limit of 4096 bytes. If this URL has been parsed ("cooked"), it is logged with local code page conversion, and is treated as a Unicode field. If this URL has not been parsed ("cooked") at the time of logging, it is copied exactly, without any Unicode conversion. If the HTTP API cannot parse this URL, a hyphen (0x002D) is used as a placeholder for the empty field.
Protocol Status	The protocol status cannot be greater than 999. If the protocol status of the response to a request is available, it is logged in this field. If the protocol status is not available, a hyphen (0x002D) is used as a placeholder for the empty field.
SiteId	Not used in this version of the HTTP API. A placeholder hyphen (0x002D) always appears in this field.
Reason Phrase	This field contains a string that identifies the type of error that is being logged. This field is never left empty.
Queue Name	This the request queue name.

Monitoring Web Site and Application Pools

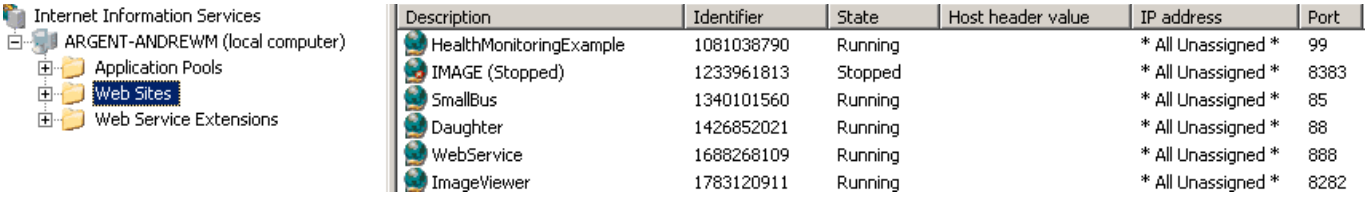
A System WMI rule (See Appendix A) can be used to check the status of the Application Pools – if the state of the pools looks like the following then the rule will generate an Alert on the stopped Pool



Rule Broken Time: 12 Aug 2010 16:50:52

image	Status = NOT UP
Argent_Web_Application_Pool	Status = UP
BusinessManagerPool	Status = UP
DefaultAppPool	Status = UP
ReportServer	Status = UP

A System WMI rule (See Appendix A) can be used to check the status of the Web Sites – if the state of the Web Sites looks like the following then the rule will generate an Alert for the Web Site that is stopped



Rule Broken Time: 12 Aug 2010 16:53:44

IMAGE(W3SVC/1233961813) is NOT ONLINE
HealthMonitoringExample(W3SVC/1081038790) is ONLINE
SmallBus(W3SVC/1340101560) is ONLINE
Daughter(W3SVC/1426852021) is ONLINE
WebService(W3SVC/1688268109) is ONLINE
ImageViewer(W3SVC/1783120911) is ONLINE

Appendix A (Application Pool State System WMI Rule)

```
Const numspace = “30”

Call enumAppPools(TargetServer)

‘ This will enumerate the app pools

Sub enumAppPools(TargetServer)
On Error Resume Next

Dim apool,obj
set obj = GetObject(“IIS://” & TargetServer & “/W3SVC/apppools”)

For each apool in obj
       Call apppoolStatus(TargetServer,apool.name)
Next

set obj = nothing
End Sub

‘ This will get the status of the app pool

Function apppoolStatus(TargetServer,apppool)
Dim obj
set obj = GetObject(“IIS://” & TargetServer & “/W3SVC/apppools/” & apppool)

if obj.apppoolstate <> 2 then
FireAlert apppool & space(numspace-(len(apppool))) & “ Status = NOT UP”, apppool
Else
WriteStatus apppool & space(numspace-(len(apppool))) & “ Status = UP”
End If

set obj = nothing
End Function

Set objSWems  = GetObject(“winmgmts:” & “{impersonationLevel=impersonate}!\\” &
TargetServer & “\root\MicrosoftIISv2”)
Set objwebstate = objSWems.ExecQuery(“Select * From IIsWebServer”)

For Each obj in objwebstate

       Set objwebname  = objSWems.ExecQuery(“Select * From IIsWebServerSetting
where name=’” & obj.name & “’”)

For Each objs in objwebname

              WebsiteName = objs.servercomment
Next

If obj.serverstate = 4 then

              FireAlert WebSiteName & “(“ & obj.name & “) is NOT ONLINE”, obj.
name
Else
              WriteStatus WebSiteName & “(“ & obj.name & “) is ONLINE”
End If

Next
```

Appendix A (Web Site State System WMI Rule)

```
Set objSWems  = GetObject(“winmgmts:” & “{impersonationLevel=impersonate}!\\” &
TargetServer & “\root\MicrosoftIISv2”)
Set objwebstate = objSWems.ExecQuery(“Select * From IIsWebServer”)

For Each obj in objwebstate

\Set objwebname  = objSWems.ExecQuery(“Select * From IIsWebServerSetting where
name=’” & obj.name & “’”)

For Each objs in objwebname

              WebsiteName = objs.servercomment
Next

If obj.serverstate = 4 then

              FireAlert WebSiteName & “(“ & obj.name & “) is NOT ONLINE”, obj.
name
Else
              WriteStatus WebSiteName & “(“ & obj.name & “) is ONLINE”
End If

Next
```

Appendix B (Performance Counter Details)

ASP.NET Applications Counters

The performance counters in this category are reset to 0 when either the application domain or Web service is restarted.

- **Cache Total Entries**. The current number of entries in the cache (both User and Internal). Internally, ASP.NET uses the cache to store objects that are expensive to create, including configuration objects, preserved assembly entries, paths mapped by the **MapPath** method, and in-process session state objects.

  Note: The "Cache Total" family of performance counters is useful for diagnosing issues with in-process session state. Storing too many objects in the cache is often the cause of memory leaks.

- **Cache Total Hit Ratio**. The total hit-to-miss ratio of all cache requests (both user and internal).

- **Cache Total Turnover Rate**. The number of additions and removals to the cache per second (both user and internal). A high turnover rate indicates that items are being quickly added and removed, which can be expensive.

- **Cache API Entries**. The number of entries currently in the user cache.

- **Cache API Hit Ratio**. The total hit-to-miss ratio of User Cache requests.

- **Cache API Turnover Rate**. The number of additions and removals to the user cache per second. A high turnover rate indicates that items are being quickly added and removed, which can be expensive.

- **Output Cache Entries**. The number of entries currently in the Output Cache.

- **Output Cache Hit Ratio**. The total hit-to-miss ratio of Output Cache requests.

- **Output Cache Turnover Rate**. The number of additions and removals to the output cache per second. A high turnover rate indicates that items are being quickly added and removed, which can be expensive.

- **Pipeline Instance Count**. The number of active pipeline instances. Only one thread of execution can be running within a pipeline instance, so this number gives the maximum number of concurrent requests that are being processed for a given application. The number of pipeline instances should be steady. Sudden increases are indicative of backend latency (see the description of Requests Rejected above).

- **Compilations Total**. The number of ASAX, ASCX, ASHX, ASPX, or ASMX files that have been compiled. This is the number of files compiled, not the number of generated assemblies. Assemblies are preserved to disk and reused until either the create time, last write time, or length of a file dependency changes. The dependencies of an ASPX page include global.asax, web.config, machine.config, dependent assemblies in the bin folder, and ASCX files referenced by the page. If you restart the application without modifying any of the file dependencies, the preserved assembly will be reloaded without requiring any compilation. This performance counter will increment only when a file is initially parsed and compiled into an assembly.

  By default, batch compilation is enabled, however, this counter will increment once for each file that is parsed and compiled into an assembly, regardless of how many assemblies are created.

- **Errors During Preprocessing**. The total number of configuration and parsing errors. This counter is incremented each time a configuration error or parsing error occurs. Even though configuration errors are cached, the counter increments each time the error occurs.

  Note: Do not rely solely upon the "Errors" performance counters to determine whether the server is healthy. They are reset to zero when the AppDomain is unloaded. They can, however, be used to dig deeper into a specific issue. In general, use the **Application_Error** event in order to alert administrators to problems.

- **Errors During Compilation**. The total number of compilation errors. The response is cached, and this counter increments only once until recompilation is forced by a file change. Implement custom error handling to raise an event.
- **Errors During Execution**. The total number of run-time errors.
- **Errors Unhandled During Execution**. The total number of unhandled exceptions at run time. This does not include the following:
    a. Errors cleared by an event handler (for example, by **Page_Error** or **Application_Error**).
    b. Errors handled by a redirect page.
    c. Errors that occur within a try/catch block.
- **Errors Unhandled During Execution/sec**. The total number of unhandled exceptions per second at run time.
- **Errors Total**. The sum of Errors During Preprocessing, Errors During Compilation, and Errors During Execution.
- **Errors Total/sec**. The total of Errors During Preprocessing, Errors During Compilation, and Errors During Execution per second.
- **Requests Executing**. The number of requests currently executing. This counter is incremented when the **HttpRuntime** begins to process the request and is decremented after the **HttpRuntime** finishes the request.
- **Requests In Application Queue**. The number of requests in the application request queue (see description of Requests Queued above). In addition to Requests Current, Requests in Application Queue provides a warning for when requests will be rejected. If there are only a couple virtual directories, increasing the default **appRequestQueueLimit** to 200 or 300 may be suitable, especially for slow applications under heavy load.
- **Requests Not Found**. The number of requests for resources not found.
- **Requests Not Authorized**. The number of request failed due to unauthorized access.
- **Requests Timed Out**. The number of requests that have timed out.
- **Requests Succeeded**. The number of requests that have executed successfully.
- **Requests Total**. The number of requests since the application was started.
- **Requests/Sec**. The number of requests executed per second. I prefer "Web Service\ISAPI Extension Requests/sec" because it is not affected by application restarts.
- **Virtual Bytes**. The current size, in bytes, of the virtual address space for this process.

The virtual address space limit of a user mode process is 2 GB, unless 3 GB address space is enabled by using the /3GB switch in boot.ini. Performance degrades as this limit is approached, and typically results in a process or system crash. The address space becomes fragmented as the 2 GB or 3 GB limit is approached, and so I recommend a conservative threshold of 1.4 or 2.4 GB, respectively. If you're running into issues here, you will see **System.OutOfMemoryException** being thrown, and this may or may not crash the process.

When running on IIS 6.0, a virtual memory limit can be set in IIS Manager. However, setting this improperly can cause problems for ASP.NET. ASP.NET expunges items from the cache to avoid exceeding the Private Bytes limit, but the algorithm uses Private Bytes and the Private Bytes limit in this determination. It does not monitor Virtual Bytes or the Virtual Bytes limit. Given that the difference between Virtual Bytes and Private Bytes is typically no more than 600 MB, you could set the Virtual Bytes limit to a value 600 MB larger than the Private Bytes limit if you are concerned about the possibility of virtual memory leaks or fragmentation. If this is desirable, set a limit for **Maximum virtual memory (in megabytes)**, found on the **Recycling** tab for the **Properties** of the application pool.

Threshold: 600 MB less than the size of the virtual address space; either 1.4 or 2.4 GB.

Web Service Counters

- **Current Connections**. A threshold for this counter is dependent upon many variables, such as the type of requests (ISAPI, CGI, static HTML, and so on), CPU utilization, and so on. A threshold should be developed through experience.

Performance Rule Is Broken If
Current Connections of Web Service (Argent_Web) GREATER THAN 200.00

Set to an acceptable number of sessions suitable for the monitored applications.

- **Total Method Requests/sec**. Used primarily as a metric for diagnosing performance issues. It can be interesting to compare this with "ASP.NET Applications\Requests/sec" and "Web Service\ISAPI Extension Requests/sec" in order to see the percentage of static pages served versus pages rendered by aspnet_isapi.dll.
- **ISAPI Extension Requests/sec**. Used primarily as a metric for diagnosing performance issues. It can be interesting to compare this with "ASP.NET Applications\Requests/sec" and "Web Service\ Total Method Requests/sec." Note that this includes requests to all ISAPI extensions, not just aspnet_isapi.dll.

Appendix C (HTTP API Reason Phrases)

Reason Phrase	Description
AppOffline	A service unavailable error occurred (an HTTP error 503). The service is not available because application errors caused the application to be taken offline.
AppPoolTimer	A service unavailable error occurred (an HTTP error 503). The service is not available because the application pool process is too busy to handle the request.
AppShutdown	A service unavailable error occurred (an HTTP error 503). The service is not available because the application shut down automatically in response to administrator policy.
BadRequest	A parse error occurred while processing a request.
Connection_Abandoned_By_AppPool	A worker process from the application pool has quit unexpectedly or orphaned a pending request by closing its handle.
Connection_Abandoned_By_ReqQueue	A worker process from the application pool has quit unexpectedly or orphaned a pending request by closing its handle. Specific to Windows Vista and Windows Server 2008.
Connection_Dropped	The connection between the client and the server was closed before the server could send its final response packet. The most common cause of this behavior is that the client prematurely closes its connection to the server.
Connection_Dropped_List_Full	The list of dropped connections between clients and the server is full. Specific to Windows Vista and Windows Server 2008.
ConnLimit	A service unavailable error occurred (an HTTP error 503). The service is not available because the site level connection limit has been reached or exceeded.
Connections_Refused	The kernel NonPagedPool memory has dropped below 20MB and http.sys has stopped receiving new connections
Disabled	A service unavailable error occurred (an HTTP error 503). The service is not available because an administrator has taken the application offline.
EntityTooLarge	An entity exceeded the maximum size that is permitted.
FieldLength	A field length limit was exceeded.
Forbidden	A forbidden element or sequence was encountered while parsing.
Header	A parse error occurred in a header.
Hostname	A parse error occurred while processing a Hostname.
Internal	An internal server error occurred (an HTTP error 500).
Invalid_CR/LF	An illegal carriage return or line feed occurred.
LengthRequired	A required length value was missing.
N/A	A service unavailable error occurred (an HTTP error 503). The service is not available because an internal error (such as a memory allocation failure) occurred.
N/I	A not-implemented error occurred (an HTTP error 501), or a service unavailable error occurred (an HTTP error 503) because of an unknown transfer encoding.

Reason Phrase	Description
Number	A parse error occurred while processing a number.
Precondition	A required precondition was missing.
QueueFull	A service unavailable error occurred (an HTTP error 503). The service is not available because the application request queue is full.
RequestLength	A request length limit was exceeded.
Timer_AppPool	The connection expired because a request waited too long in an application pool queue for a server application to dequeue and process it. This timeout duration is **ConnectionTimeout**. By default, this value is set to two minutes.
Timer_ConnectionIdle	The connection expired and remains idle. The default **ConnectionTimeout** duration is two minutes.
Timer_EntityBody	The connection expired before the request entity body arrived. When it is clear that a request has an entity body, the HTTP API turns on the **Timer_EntityBody** timer. Initially, the limit of this timer is set to the **ConnectionTimeout** value (typically 2 minutes). Each time another data indication is received on this request, the HTTP API resets the timer to give the connection two more minutes (or whatever is specified in **ConnectionTimeout**).
Timer_HeaderWait	The connection expired because the header parsing for a request took more time than the default limit of two minutes.
Timer_MinBytesPerSec-ond	The connection expired because the client was not receiving a response at a reasonable speed. The response send rate was slower than the default of 240 bytes/sec.
Timer_ReqQueue	The connection expired because a request waited too long in an application pool queue for a server application to dequeue. This timeout duration is **ConnectionTimeout**. By default, this value is set to two minutes. Specific to Windows Vista and Windows Server 2008.
Timer_Response	Reserved. Not currently used.
URL	A parse error occurred while processing a URL.
URL_Length	A URL exceeded the maximum permitted size.
Verb	A parse error occurred while processing a verb.
Version_N/S	A version-not-supported error occurred (an HTTP error 505).