Monitoring Web Servers

eG Enterprise v6

Restricted Rights Legend

The information contained in this document is confidential and subject to change without notice. No part of this document may be reproduced or disclosed to others without the prior permission of eG Innovations Inc. eG Innovations Inc. makes no warranty of any kind with regard to the software and documentation, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose.

Trademarks

Microsoft Windows, Windows 2008, Windows 2012, Windows 7, Windows 8 and Windows 10 are either registered trademarks or trademarks of Microsoft Corporation in United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Copyright

Table of Contents

Table of Figures

Chapter 1

Introduction

Web servers like Microsoft IIS, Apache, and Sun ONE web servers are the heart of IT infrastructures in various domains - Healthcare, Banking, Trading, Logistics, etc. To ensure scalability and high performance, most web sites are being architected to use the multi-tier model - i.e., with the web server (IIS, Apache, etc.) functioning as the front-end, the middleware application server (J2EE, .Net based, etc.) that hosts the business logic functioning as the mid-tier, and a database server (SQL, Oracle, etc.) as the backend. In such architectures, the web server plays a pivotal role since all users to the other tiers are routed via the web server and hence, any slowdown or problem in the web server tier can adversely impact the end user experience.

The availability of a web site and the response time for user accesses to the site are the most critical metrics of web performance. Both these metrics may vary depending from one website to another and even from one transaction to another. For instance, one set of application components may come into play when a user logs in to an eBanking site, while a set of components may be invoked when a user transfers funds between his/her accounts. Consequently, a web monitoring solution must be able to report the availability and response time for individual user transactions to a web site.

Most web monitoring solutions rely on request emulation to monitor web transactions to a site. These request emulators generate synthetic requests periodically from one or more locations to the site and monitor the availability and response time for each transaction. This simple yet elegant solution provides the external perspective of the site.

The main limitations of a request emulation-only approach are:

a. This approach cannot be used to monitor the most critical transactions to a web site e.g., a user making a payment, a user registering to a web site, etc.,

b. Moreover, this approach mainly samples the functioning of the target environment. If a specific transaction is failing, say 10% of the time, the emulation approach only has a 10% chance of reporting the problem. Consequently, this approach is able to consistently detect and report problems only when they are severe enough to impact the end user performance. i.e., a request emulated approach only enables reactive monitoring.

The eG web monitor adopts a unique two-pronged approach for web transaction monitoring. The external agent uses request emulation to assess the user experience from different locations. By doing so, the external agent captures the effect of the network latency and the server-side processing time on the end user experience.

To quantify the server processing times for real user requests (not emulated requests), the eG internal agent deploys a proprietary web-adapter technology. This technology enhances web servers with the capability to track HTTP/HTTPS requests to a web server and the corresponding responses. For each transaction that is configured for monitoring, the web adapter analyzes the request URLs and responses to report various metrics relating to individual web transactions in real-time.

The monitoring is done in an implementation-independent manner, as a result of which eG agents are able to monitor Java (Servlets, EJB, JSPs) and other non-Java implementations (ASP, PHP, CGI, etc.) with equal felicity.

INTRODUCTION

Since it is able to monitor real-user transactions to web servers in real-time, eG Enterprise's web adapter technology enables the agents to proactively monitor and quantify all anomalies that may occur in a web infrastructure. The ability to offer real-time monitoring of 100% of the real user transactions, without the need for explicit, expensive logging is a key distinguishing feature of eG Enterprise's Web Server Monitor. This one-of-a-kind monitoring capacity supports Microsoft IIS, Apache, Sun ONE and other popular web servers.

eG Enterprise offers specialized monitoring models for each of these popular web servers. The statistics reported by these models, enable administrators to find accurate answers to the following performance queries:

External monitoring	• Is the web site available for user accesses from different locations? • What is the response time for user accesses to the site from different geographic locations? • Is a slowdown due to increased network latency or due to increased server-side processing?
Internal transaction monitoring	• How are the critical transactions of a web site functioning? • What is the request rate for each transaction? • What is the average response time for each transaction? • Are there many aborts for the transaction?
Web site monitoring	• What is the status of the different web sites hosted on a single web server? • Are there many errors occurring in the system? • Are the servers supporting the web infrastructure adequately sized? • Are there usage trends that need to be accounted for future capacity planning?
Bottleneck detection	Is an increase in server-side processing time due to the web server or due to the middleware application server or due to the database?
Capacity planning	Is the load being effectively balanced across all the web servers? Are the critical web server processes up and running?

This document engages you in an elaborate discussion on how eG Enterprise monitors each of the popular web servers in the market.

Chapter 2

Monitoring the Apache, iPlanet, and IBM HTTP Servers

iPlanet Web Server is an extremely powerful multi-process, multi-threaded, secure web server built on open standards that enables your business enterprise to seamlessly integrate with other internal and external systems. IBM HTTP Server (IHS), on the other hand, is a web server based on the Apache Software Foundation's Apache HTTP Server that runs on AIX, HP-UX, Linux, Solaris, Windows NT, and z/OS.

This chapter discusses the monitoring model that eG Enterprise prescribes for monitoring the above-mentioned web servers, and the statistics that the eG agent pulls out from these web servers.

eG Enterprise offers a common *Web* server model (see Figure 2.1) to monitor the iPlanet/IBM HTTP/Apache web servers. The **Operating System**, **Network**, and **Tcp** layers have already been discussed in the *Monitoring Unix and Windows Servers* document. The **Application Processes** layer for the *Web* server is mapped to an additional **SSL Certificate Validity** test, which will be discussed later in this chapter. Above the **Application Processe** layer is the **Web Server** layer. This layer captures the state of the web server itself – whether the server is responding to user requests or not, whether the response is timely, whether the server is under overload, etc. Since a single web server may handle multiple web sites (e.g., in a virtual hosting scenario), the next layer is the **Web Site** layer that captures the health of a specific web site. While the overall health of a web site is represented by the **Web Site** layer, the status of the different key transactions of a web site are represented by the **Web Transactions** layer.

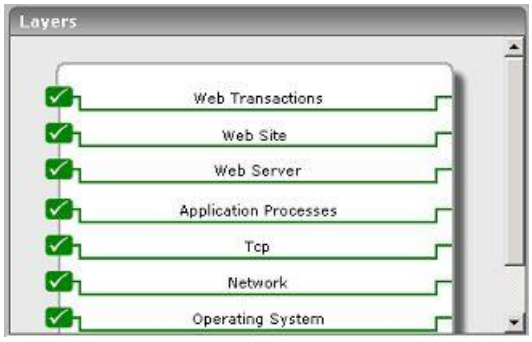


Figure 2.1: The different layers that the eG Enterprise suite monitors for a web server

To monitor the health of web servers, sites, and transactions, eG agents use the unique web adapter capability that allows a wealth of information to be collected from web servers without requiring explicit, expensive log files. More details of the web adapter technology can be found in the **EG USER MANUALS** and the technical white papers**.**

The following sections delve into the details of the tests that correspond to each of the layers in Figure 2.1.

2.1 The Application Processes Layer

This layer depicts the states of the different processes that must be executing for the Web server to be available.

Figure 2.2: The tests mapped to the Application Processes layer

In addition, the layer can also be optionally configured to monitor the validity of the SSL certificates (if any) that may have been installed on the Web server. For this, an **SSL Certificate Validity** test will have to be enabled. The details of this test have been provided below.

2.1.1 SSL Certificate Test

This test reports how long (in days) the SSL certificates that have been configured for monitoring will remain valid.

By default, this test is disabled. To enable the test, follow the *Agents -> Tests -> Enable/Disable* menu sequence, select the **Component type**, pick *Performance* as the **Test type**, select this test from the **DISABLED TESTS** list, and click the **<<** button to enable it.

Purpose	Reports how long (in days) the SSL certificates that have been configured for monitoring will remain valid
Target	A web server
Agent deploying this test	An internal agent
Configurable parameters for this test	1. **TEST PERIOD** – How often should the test be executed 2. **HOST** - The host for which this test is being configured 3. **TARGETS** - If you want to monitor specific SSL-enabled web sites, then, provide a comma-separated list of *{HostIP/Name}:{Port)* pairs, which represent the web sites to be monitored. For example, *192.168.10.7:443,192.168.10.8:443*. The test connects to each IP/port pair and checks for the validity of the certificate associated with that target. One set of metrics is reported for each target. The descriptor represents the common name

	(CN) value of the SSL certificate. By default, this parameter is set to the *<IP_of_the_monitored_web/application_server>:<Port_on_which_the_server_listens>*. If you do not want to monitor the validity of certificates based on configured **TARGETS**, set this parameter to *none*. 4. **TARGETFILES** – To monitor specific certificate files, provide a comma-separated list of file paths for the SSL certificates that are to be monitored in the **TARGETFILES** text box. For example, *C:|server.crt, D:|admin.crt*. The test reads the SSL Certificates for the web sites that are to be monitored from this location and checks for the validity. If you do not want to check the validity of specific certificate files, set this parameter to *none*. 5. **KEYSTORE FILE** - A keystore is a database (usually a file) that can contain trusted certificates and combinations of private keys with their corresponding certificates. If you are looking to monitor the certificates contained within a keystore file, then provide the full path to this file in the **KEYSTORE FILE** text box. For example, the location of this file may be: *C:|egurkha|manager|tomcat|webapps|eGmanager.bin*. In this case, the test automatically accesses each of the certificates that the specified keystore contains, and checks its validity. If you do not want to monitor the certificates in a keystore, set this parameter to *none*. 6. **KEYSTORE PASSWORD** - If a **KEYSTORE FILE** is provided, then, in the **KEYSTORE PASSWORD** text box, provide the password that is used to obtain the associated certificate details from the keystore file. If *none* is specified against **KEYSTORE FILE**, then, enter *none* here as well. 7. **CONFIRM PASSWORD** - Confirm the **KEYSTORE PASSWORD** by retyping it here.		
Outputs of the test	One set of outputs for every **TARGET** and/or every **TARGETFILE** and/or the unique key assigned to each certificate in the specified **KEYSTORE FILE**		
Measurements of the test	**Measurement**	**Measurement Unit**	**Interpretation**
	SSL certificate validity: Indicates the number of days from the current day for which this SSL certificate will be valid.	Number	

Note:

This test is available for the following component types:

- Web server
- SSL web server
- IIS web server
- IIS SSL server
- Sun Java web server
- Apache web server
- WebLogic server
- WebSphere server
- SunONE application server
- JRUN server
- Citrix NetScaler

2.2 The Web Server Layer

This layer tracks the health of a web server application. An external, Http test emulates a user connection to the web server, retrieves a web page from the server and measures the availability of the server and the time taken by the server to respond to the emulated user request. A second, internal WebServer test is based on eG Enterprise's web adapter capability (see Figure 2.3). The web adapter configured for a web server periodically monitors all the requests handled by the server. By snooping on user requests to the server and responses sent out by the server, the web adapter reports a variety of statistics pertaining to the web server. The WebServer test exports the information pertaining to the **Web Server** layer to the eG manager. The following sections describe the Http test and the WebServer test in detail.

Figure 2.3: The tests that map to the Web Server layer

2.2.1 HTTP Test

The details of the Http test that emulates a user accessing a web server are provided below. Since this test can be executed from a location external to the web server, this test presents an unbiased external perspective of the state of the web server.

Purpose	This test measures the state of a web server
Target	A web server
Agent deploying this test	An external agent; if you are running this test using the external agent on the eG manager box, then make sure that this external agent is able to communicate with the port on which the target Webserver is listening. Alternatively, you can deploy the external agent that will be running this test on a host that can access the port on which the target Web server is listening.
Configurable parameters for this test	1. **TEST PERIOD** – How often should the test be executed 2. **HOST** - The host for which the test is to be configured. 3. **PORT** - The port to which the specified **HOST** listens 4. **URL** – This test emulates a user accessing a specific web page(s) on the target web server to determine the availability and responsiveness of the server. To enable this emulation, you need to configure the test with the URL of the web page that it should access. Specify this URL against the **URL** parameter. If required, you can even configure multiple URLs – one each for every web page that the test should attempt to access. If each URL configured requires special permissions for logging in, then, you need to configure the test with separate credentials for logging into every URL. Likewise, you need to provide instructions to the test on how to validate the content returned by every URL, and also set an encoding format for each URL. To enable administrators to easily configure the above per URL, eG Enterprise provides a special interface. To access this interface, click on the encircled '+' button alongside the **URL** text box in the test configuration page. Alternatively, you can even click on the encircled '+' button adjacent to the **URL** parameter in the test configuration page. To know how to use this special interface, refer to Section 2.2.1.1. of this document. 5. **COOKIEFILE** – Whether any cookies being returned by the web server need to be saved locally and returned with subsequent requests 6. **PROXYHOST** – The host on which a web proxy server is running (in case a proxy server is to be used)

	7. **PROXYPORT** – The port number on which the web proxy server is listening 8. **PROXYUSERNAME** – The user name of the proxy server 9. **PROXYPASSWORD** – The password of the proxy server 10. **CONFIRM PASSWORD** – Confirm the password by retyping it here. 11. **TIMEOUT** – Here, specify the maximum duration (in seconds) for which the test will wait for a response from the server. The default **TIMEOUT** period is 30 seconds.		
Outputs of the test	One set of outputs for every URL being monitored		
Measurements of the test	**Measurement**	**Measurement Unit**	**Interpretation**
	Availability: This measurement indicates whether the server was able to respond successfully to the query made by the test.	Percent	Availability failures could be caused by several factors such as the web server process(es) being down, the web server being misconfigured, a network failure, etc. Temporary unavailability may also occur if the web server is overloaded. Availability is determined based on the response code returned by the server. A response code between 200 to 300 indicates that the server is available.
	Total response time: This measurement indicates the time taken by the server to respond to the requests it receives.	Secs	Response time being high denotes a problem. Poor response times may be due to the server being overloaded or misconfigured. If the URL accessed involves the generation of dynamic content by the server, backend problems (e.g., an overload at the application server or a database failure) can also result in an increase in response time.
	TCP connection availability: This measure indicates whether the test managed to establish a TCP connection to the server.	Percent	Failure to establish a TCP connection may imply that either the web server process is not up, or that the process is not operating correctly. In some cases of extreme overload, the failure to establish a TCP connection may be a transient condition. As the load subsides, the server may start functioning properly again.
	TCP connect time: This measure quantifies the time for establishing a TCP connection to the web server host.	Secs	Typically, the TCP connection establishment must be very small (of the order of a few milliseconds). Since TCP connection establishment is handled at the OS-level, rather than by the application, an increase in this value signifies a system-level bottleneck on the host that supports the web server.

	Server response time: This measure indicates the time period between when the connection was established and when the server sent back a HTTP response header to the client.	Secs	While the total response time may depend on several factors, the server response time is typically, a very good indicator of a server bottleneck (e.g., because all the available server threads or processes are in use).
	Response code: The response code returned by the server for the simulated request	Number	A value between 200 and 300 indicates a good response. A 4xx value indicates a problem with the requested content (eg., page not found). A 5xx value indicates a server error.
	Content length: The size of the content returned by the server	Kbytes	Typically the content length returned by the server for a specific URL should be the same across time. Any change in this metric may indicate the need for further investigation on the server side.
	Content validity: This measure validates whether the server was successful in executing the request made to it.	Percent	A value of 100% indicates that the content returned by the test is valid. A value of 0% indicates that the content may not be valid. This capability for content validation is especially important for multi-tier web applications. For example, a user may not be able to login to the web site but the server may reply back with a valid HTML page where in the error message, say, "Invalid Login" is reported. In this case, the availability will be 100 % (since we got a valid HTML response). If the test is configured such that the content parameter should exclude the string "Invalid Login," in the above scenario content validity would have a value 0.

	DNS availability: Indicates whether the DNS server was able to respond successfully to the request made to it.	Percent	While the value 100 for this measure indicates that the DNS server is available and successfully responded to the request, the value 0 indicates that the DNS server is unavailable or is not responding to requests. Availability failures could be caused by many reasons such as a network failure. Sometimes, the DNS server may be reachable through basic network testing, but may not respond to DNS queries from clients. **Note:** This measure will be able to report a value only if the **URL** parameter of the test is configured with a domain name-based URL – eg., http://www.eginnovations.com, http://www.eBooks.com. If the **URL** parameter is configured with an IP-based URL instead – eg., http://192.168.10.21:80, http://192.168.10.34:7077 – then, this measure will not report any value. This is because, to report the availability of the DNS server, the test attempts to connect to the DNS server and resolve the domain name in the **URL** to its IP address. If the test is able to perform domain name – IP address resolution successfully, it reports the value *100* for this measure. If the resolution fails, the test reports the value *0*. In case of an IP-based URL naturally, the test will not be able to find any domain name to resolve. The test therefore will not report any value for this measure in that case.
	Data transfer time: Indicates the time taken for a data transfer between the drive and the host system.	Secs	Data transfer time being high denotes a problem.

2.2.1.1 Configuring Multiple URLs for Monitoring

By default, the HTTP test will be configured with the URL of the home page of the web server being monitored. To configure additional URLs, do the following:

1. Click on the encircled '+' button alongside the **URL** text box in Figure 2.4.

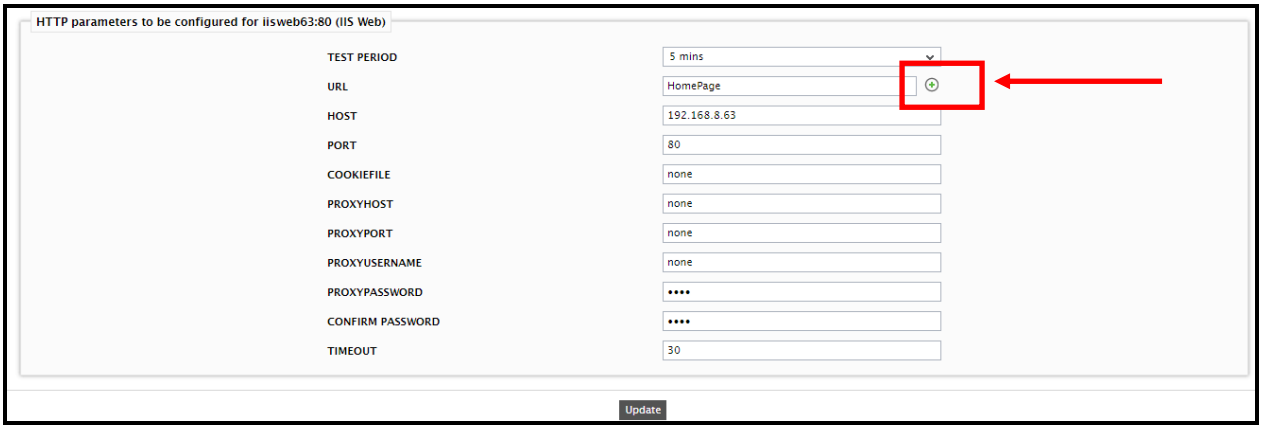


Figure 2.4: Configuring the HTTP test

2. Figure 2.5 then appears. To add another URL, click the **Add More** button in Figure 2.5.

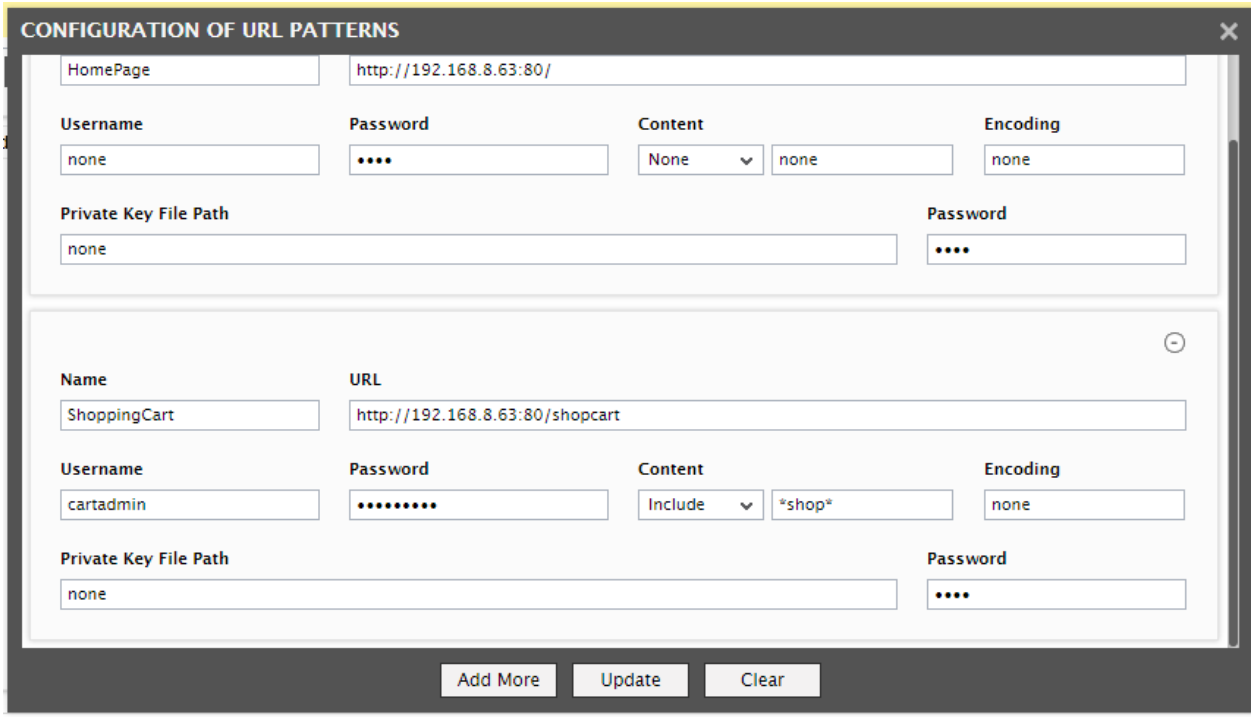


Figure 2.5: Configuring multiple URLs

3. Another URL specification section will appear. Specify the following in that section:

- **Name**: Specify a unique name by which the URL you will be specifiying shortly will be referred to across the eG user interface. This is the name that will appear as the descriptor of this test.
- **URL**: Enter the URL of the web page that this test should access.
- **Username** and **Password**: These parameters are to be set only if a specific user name / password has to be specified to login to the web page (i.e., **URL**) that you have configured for monitoring. In this case, provide valid login credentials using the **Username** and **Password** text boxes. If the web server on which **HTTP** test

executes supports 'Anonymous user access', then these parameters will take either of the following values:

- A valid **Username** and **Password** for the configured **URL**
- *none* in both the **Username** and **Password** text boxes of the configured **URL**, if no user authorization is required
- Some web servers however, support NTLM (Integrated Windows) authentication, where valid login credentials are mandatory. In other words, a *none* specification will not be supported by such web servers. Therefore, in this case, against each configured **URL**, you will have to provide a valid **Username** in the format: *domainname*|*username,* followed by a valid **Password**.
- Please be sure to check if your web site requires HTTP authentication while configuring this parameter. HTTP authentication typically involves a separate pop-up window when you try to access the page. Many sites use HTTP POST for obtaining the user name and password and validating the user login. In such cases, the username and password have to be provided as part of the POST information and NOT as part of the **CREDENTIALS** specification for the HTTP test.
- **Content**: The **Content** parameter has to be configured with an instruction:valu**e** pair that will be used to validate the content being returned by the test. If the **Content** value is *None*, no validation is performed. On the other hand, if you pick the *Include* option from the **Content** list, it indicates to the test that for the content returned by the web server to be valid, the content must include the specified value (a simple string search is done in this case). This value should be specified in the adjacent text box. Similarly, if the *Exclude* option is chosen from the **Content** drop-down, it indicates to the test that the server's output is valid if it does not contain the value specified in the adjacent text box. The *Include* or *Exclude* value you specify in the text box can include wildcard characters. For example, an *Include* instruction can be **Home page**.
- **Encoding**: Sometimes the eG agent has to parse the **URL** content with specific encoding other than the default (ISO-8859-1) encoding. In such a case, specify the type of encoding using which the eG agent can parse the **URL** content in the **Encoding** text box. By default, this value is *none*.
- **Private Key File Path** and **Password**: SSL-enabled web sites are typically secured by a private key, public key, or a public-private key pair. If the web page configured for this test is SSL-enabled – i.e., if an HTTPS URL is specified against **URL** – and the contents of this web page can only be accessed using a **private key**, then the full path to the private key file will have to be provided against **Private key file path** and the password of the private key file should be specified against **Password**. If no such private key protects the contents of the configured **URL**, then set the **Private key file path** and its **Password** to *none*.

4. Similarly, you can add multiple URL specifications. To remove a URL specification, click on the encircled '-' button corresponding to it. To clear all URL specifications, click the **Clear** button in Figure 2.5. To update all the changes you made, click the **Update** button.
5. Once **Update** is clicked, you will return to the test configuration page (see Figure 2.6). The **URL** text box in the test configuration page will display just the **Name**s – i.e., the unique display names – that you may have configured for the multiple URLs, as a comma-separated list. To view the complete URL specification, click the encircled '+' button alongside the **URL** text box, once again.

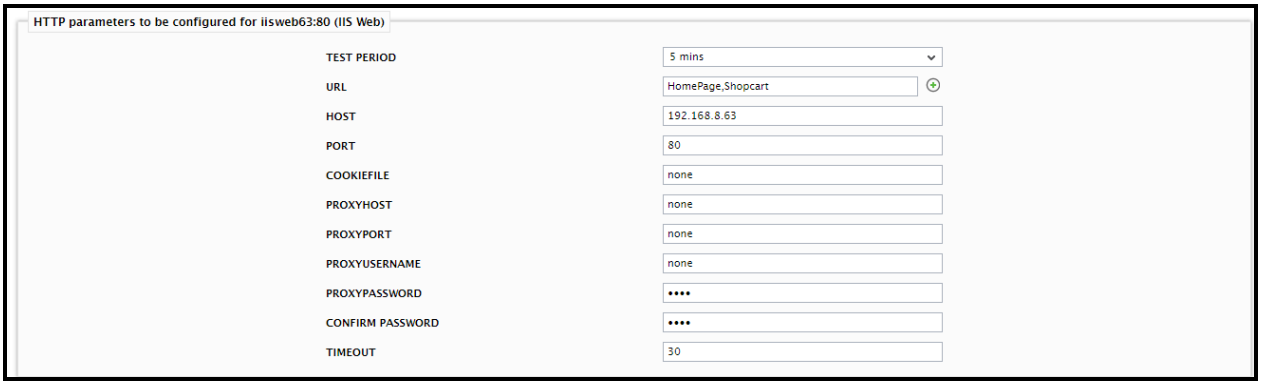


Figure 2.6: The URL text box displaying only the display names of the configured URLs as a comma-separated list

2.2.2 Web Server Test

This internal test complements the measurements made from an external perspective by the Http test. In order to collect various statistics pertaining to the availability, performance, and usage of a web server, this test interfaces with the eG web adapter. The statistics collected by this test are detailed below:

Purpose	To measure the state of a web server		
Target of the test	A web server instance		
Agent deploying the test	An internal agent		
Configurable parameters for the test	1. **TEST PERIOD** - How often should the test be executed 2. **HOST -** The host for which the test is to be configured 3. **PORT -** the port to which the specified **HOST** listens		
Outputs of the test	One set of results for every web server monitored		
Measurements made by the test	**Measurement**	**Measurement Unit**	**Interpretation**
	Connections: Rate of connections to the web server.	Conns/Sec	An increase or decrease in connection rate can represent a change in user workload
	Requests: Rate of requests to the web server during the last measurement period.	Reqs/Sec	With the advent of HTTP/1.1, multiple requests can be transmitted over the same TCP connection. The ratio of requests per connection can provide an idea of the effectiveness of the HTTP 1.1 protocol.

	Data transmitted: Rate at which the data was transmitted by the server during the last measurement period.	KB/Sec	A large increase in the data transmission rate can be indicative of an increase in the popularity of one or more web sites hosted on the server.
	Data received: Rate at which the data was received by the server during the last measurement period.	KB/Sec	An increase in this value is indicative of an increase in user requests to the server.
	Errors: Percentage of error responses from the server during the last measurement period.	Percent	Percentage of responses with a 400 or 500 status code.
	Aborts: Percentage of requests aborted by users (a request is deemed to have been aborted if either the connection is closed without any response being generated, or if the TCP connection is closed as the server is reading the request or as it is responding to the request) .	Percent	A high percentage of aborts can happen if the web server's responsiveness has dramatically reduced. Aborts may also occur because of backend errors (e.g., application failures, database connection problems etc.).
	300 responses: Percentage of responses with a status code in the 300-399 range during the last measurement period.	Percent	300 responses could indicate page caching on the client browsers. Alternatively 300 responses could also indicate redirection of requests. A sudden change in this value could indicate a problem condition.
	400 errors: Percentage of responses with a status code in the range 400-499 during the last measurement period.	Percent	A high value indicates a number of missing/error pages.
	500 rrors: Percentage of responses with a status code in the range 500-599 during the last measurement period.	Percent	Since responses with a status code of 500-600 indicate server side processing errors, a high value reflects an error condition.
	Current requests: Number of server threads/processes currently in use for serving requests (this measurement is not available for Apache web servers).	Number	If a majority of the server threads/processes are in use simultaneously, this may be indicative of a server bottleneck.

Note:

The following measures are not available in the Windows version of the product:

- Aborts
- 300 responses
- 500 errors

Note:

The WebServer test will report metrics for an IIS web server executing on a Windows 2008 host, only if the **IIS Management Scripts and Tools** feature is installed on that host, and the **Web Server** role you create enables this feature. If the aforesaid feature is not enabled for the **Web Server** role, then remove the role and re-create it with the feature enabled.

2.2.3 HTTP Posts Test

The details of the HttpPost test are provided below. Since this test can be executed from a location external to the web server, this test presents an unbiased external perspective of the state of the web server. This test uses the POST command to submit its parameters. It is an optional test and can be configured only when the check box alongside it in the **DISABLED TESTS** list of the **AGENTS – TESTS CONFIGURATION** page is clicked. By default, this check box is deselected.

Purpose	This test measures the state of a web server
Target	A web server
Agent deploying this test	An external agent; if you are running this test using the external agent on the eG manager box, then make sure that this external agent is able to communicate with the port on which the target Webserver is listening. Alternatively, you can deploy the external agent that will be running this test on a host that can access the port on which the target Web server is listening.
Configurable parameters for this test	1. **TEST PERIOD** – How often should the test be executed 2. **URL** – The web page being accessed. While multiple URLs (separated by commas) can be provided, each URL should be of the format **URL name:URL value**. **URL name** is a unique name assigned to the URL, and the **URL value** is the value of the URL. For example, a URL can be specified as **HomePage:http://192.168.10.12:7077/**, where **HomePage** is the **URL name** and http://192.168.10.12:7077/ is the **URL value**. 3. **HOST** - The host for which the test is to be configured. 4. **PORT** - The port to which the specified **HOST** listens 5. **COOKIEFILE** – Whether any cookies being returned by the web server need to be saved locally and returned with subsequent requests

<table>
<tr><td></td><td colspan="3">
6. PROXYHOST – The host on which a web proxy server is running (in case a proxy server is to be used)
7. PROXYPORT – The port number on which the web proxy server is listening
8. PROXYUSERNAME – The user name of the proxy server
9. PROXYPASSWORD – The password of the proxy server
10. CONFIRM PASSWORD – Confirm the password by retyping it here.
11. CONTENT – Is a set of instruction:value pairs that are used to validate the content being returned by the test. If the CONTENT value is none:none, no validation is performed. The number of pairs specified in this text box, must be equal to the number of URLs being monitored. The instruction should be one of Inc or Exc. Inc tells the test that for the content returned by the web server to be valid, the content must include the specified value (a simple string search is done in this case). An instruction of Exc instructs the test that the server's output is valid if it does not contain the specified value. In both cases, the content specification can include wild card patterns. For example, an Inc instruction can be Inc:*Home page*.
12. CREDENTIALS – The Http test supports HTTP authentication. The CREDENTIALS parameter is to be set if a specific user name / password has to be specified to login to a page. This parameter is a comma separated list of user name:password pairs, one pair for each URL being monitored. A value of none:none indicates that user authorization is not required. Please be sure to check if your web site requires HTTP authentication while configuring this parameter. HTTP authentication typically involves a separate pop-up window when you try to access the page. Many sites use HTTP POST for obtaining the user name and password and validating the user login. In such cases, the username and password have to be provided as part of the POST information and NOT as part of the CREDENTIALS specification for the Http test.
13. TIMEOUT - Here, specify the maximum duration (in seconds) for which the test will wait for a response from the server. The default TIMEOUT period is 30 seconds.
</td></tr>
<tr><td>Outputs of the test</td><td colspan="3">One set of outputs for every URL being monitored</td></tr>
<tr><td rowspan="2">Measurements of the test</td><td>Measurement</td><td>Measurement Unit</td><td>Interpretation</td></tr>
<tr><td>Availability:
This measurement indicates whether the server was able to respond successfully to the query made by the test.</td><td>Percent</td><td>Availability failures could be caused by several factors such as the web server process(es) being down, the web server being misconfigured, a network failure, etc. Temporary unavailability may also occur if the web server is overloaded. Availability is determined based on the response code returned by the server. A response code between 200 to 300 indicates that the server is available.</td></tr>
</table>

	Total response time: This measurement indicates the time taken by the server to respond to the requests it receives.	Secs	Response time being high denotes a problem. Poor response times may be due to the server being overloaded or misconfigured. If the URL accessed involves the generation of dynamic content by the server, backend problems (e.g., an overload at the application server or a database failure) can also result in an increase in response time.
	TCP connection availability: This measure indicates whether the test managed to establish a TCP connection to the server.	Percent	Failure to establish a TCP connection may imply that either the web server process is not up, or that the process is not operating correctly. In some cases of extreme overload, the failure to establish a TCP connection may be a transient condition. As the load subsides, the server may start functioning properly again.
	TCP connect time: This measure quantifies the time for establishing a TCP connection to the web server host.	Secs	Typically, the TCP connection establishment must be very small (of the order of a few milliseconds). Since TCP connection establishment is handled at the OS-level, rather than by the application, an increase in this value signifies a system-level bottleneck on the host that supports the web server.
	Server response time: This measure indicates the time period between when the connection was established and when the server sent back a HTTP response header to the client.	Secs	While the total response time may depend on several factors, the server response time is typically, a very good indicator of a server bottleneck (e.g., because all the available server threads or processes are in use).
	Response code: The response code returned by the server for the simulated request	Number	A value between 200 and 300 indicates a good response. A 4xx value indicates a problem with the requested content (eg., page not found). A 5xx value indicates a server error.
	Content length: The size of the content returned by the server	KBtes	Typically the content length returned by the server for a specific URL should be the same across time. Any change in this metric may indicate the need for further investigation on the server side.

	Content validity: This measure validates whether the server was successful in executing the request made to it.	Percent	A value of 100% indicates that the content returned by the test is valid. A value of 0% indicates that the content may not be valid. This capability for content validation is especially important for multi-tier web applications. For example, a user may not be able to login to the web site but the server may reply back with a valid HTML page where in the error message, say, "Invalid Login" is reported. In this case, the availability will be 100 % (since we got a valid HTML response). If the test is configured such that the content parameter should exclude the string "Invalid Login," in the above scenario content validity would have a value 0.

2.2.4 SiteMinder Web Access Test

To control user accesses to a web site, web servers are configured to work with SiteMinder - a platform that authenticates, authorizes, and manages web sites. Using SiteMinder, administrators can enable **Single Sign-On** (SSO) for multiple web sites operating in an environment. With the SSO property of access control, a user logs in once and gains access to multiple web sites at one go, without being prompted to log into each of them separately. While SiteMinder's SSO capability saves the time and effort required to manually sign into multiple web sites, a slowdown in SiteMinder can delay/suspend accesses to all the web sites managed (i.e., protected) by it! But should SiteMinder be blamed every time a user complaints of a slowdown when accessing a protected web site? Maybe not! Given below are the steps that occur when a user tries to access a web site protected by SiteMinder:

- The user requests for a web page using HTTP/HTTPS.
- The request is received by the web server and is intercepted by the SiteMinder web agent.
- The web agent determines whether or not the resource is protected. If the resource is protected, SiteMinder forces the user to login using their credentials. Typically, this is done via an HTTP POST request.
- SiteMinder authenticates the user and verifies whether or not the authenticated user is authorized for the requested web page, based on rules and policies specified in the Policy store.
- After the user is authenticated and authorized, SiteMinder grants access to the web page. Resource grant is done by providing a cookie to the client browser.

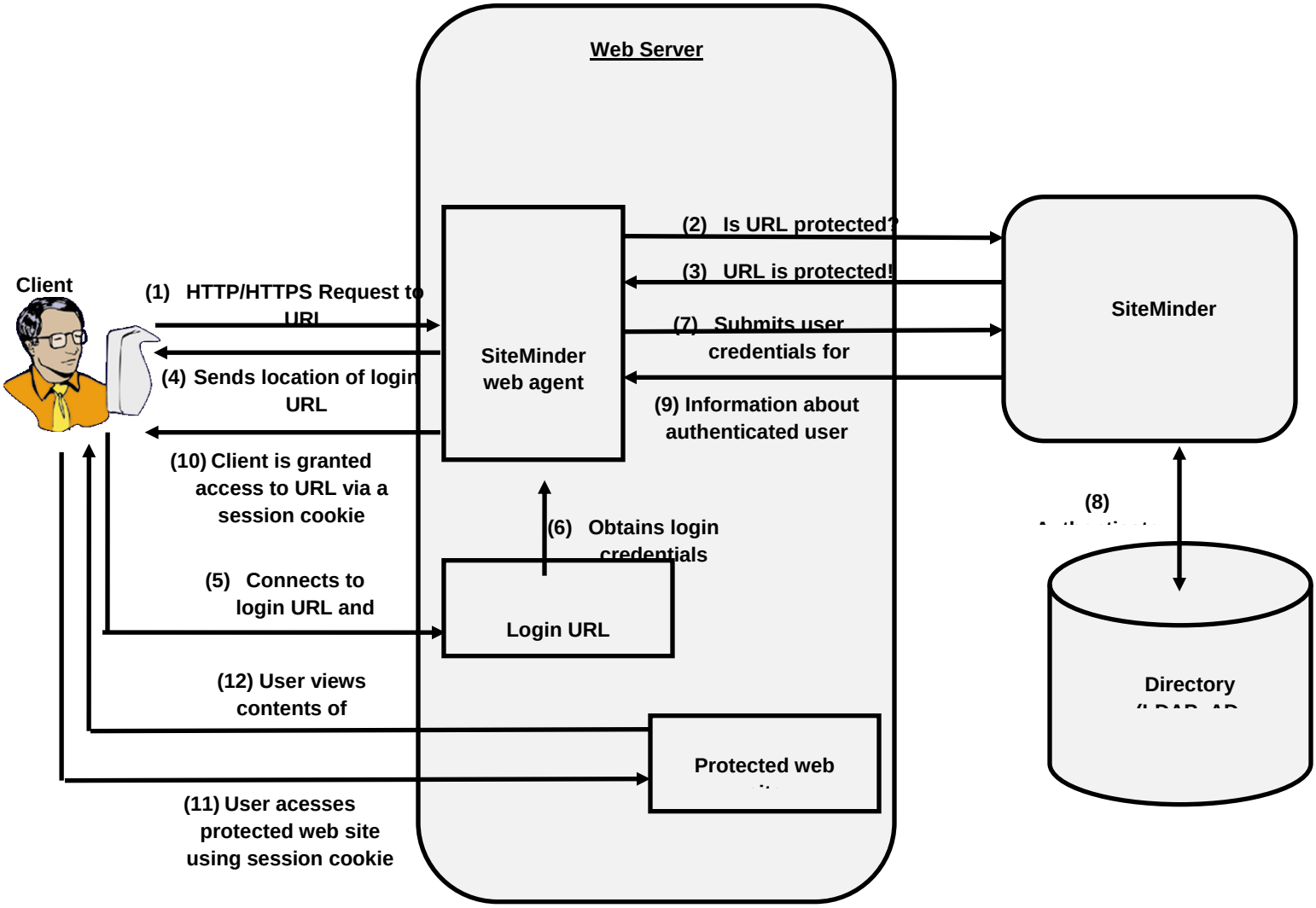


Figure 2.7: Accessing a protected web site on a web server configured to work with SiteMinder

From Figure 2.4, it is clear that a problem in any step of this multi-step user interaction can delay web site accesses! This means that a slowdown in the SiteMinder policy server can be **one of the reasons** for a delay in accessing a web site, **but it need not be the'root-cause'**! A processing bottleneck in the web server too can cause delays in web site accesses. Likewise, if the web agent takes too long to redirect the user's request to the login URL, the user experience with the site is bound to suffer. Similarly, if the web page requested is large and takes too long to load, access delays will become unavaoidable. In such circumstances, administrators will have trouble isolating the 'source' of the slowdown. If the root-cause of the slowdown is not diagnosed quickly, the problem will remain unresolved for a long time, resulting in a steep fall in service levels, increase in penalties and loss of reputation.

To avoid this, administrators will have to quickly pin-point the root-cause of a slowdown and fix it; for this, they need to know how much time each step of the user interaction with a protected web site takes. The **SiteMinder Web**

Access test does this job. The test emulates a user accessing a web site protected by SiteMinder and reports the time taken by each step of the process, so that the precise step at which the slowdown occurred can be accurately isolated and the problem promptly eliminated.

Purpose	Emulates a user accessing a web site protected by SiteMinder and reports the time taken by each step of the process, so that the precise step at which the slowdown occurred can be accurately isolated and the problem promptly eliminated
Target	A web server
Agent deploying this test	An external agent; if you are running this test using the external agent on the eG manager box, then make sure that this external agent is able to communicate with the port on which the target Webserver is listening. Alternatively, you can deploy the external agent that will be running this test on a host that can access the port on which the target Web server is listening.
Configurable parameters for this test	1. **TEST PERIOD** – How often should the test be executed 2. **HOST** - The host for which the test is to be configured. 3. **PORT** - The port to which the specified **HOST** listens 4. **URL** – The web page being accessed. While multiple URLs (separated by commas) can be provided, each URL should be of the format **URL name:URL value**. **URL name** is a unique name assigned to the URL, and the **URL value** is the value of the URL. For example, a URL can be specified as **HomePage:http://192.168.10.12:7077/**, where **HomePage** is the **URL name** and http://192.168.10.12:7077/ is the **URL value**. 5. **COOKIEFILE** – Whether any cookies being returned by the web server need to be saved locally and returned with subsequent requests 6. **PROXYHOST** – The host on which a web proxy server is running (in case a proxy server is to be used) 7. **PROXYPORT** – The port number on which the web proxy server is listening 8. **PROXYUSERNAME** – The user name of the proxy server 9. **PROXYPASSWORD** – The password of the proxy server 10. **CONFIRM PASSWORD** – Confirm the password by retyping it here. 11. **CONTENT – I**s a set of instruction:value pairs that are used to validate the content being returned by the test. If the **CONTENT** value is *none:none*, no validation is performed. The number of pairs specified in this text box, must be equal to the number of URLs being monitored. The instruction should be one of *Inc* or *Exc*. *Inc* tells the test that for the content returned by the web server to be valid, the content must include the specified value (a simple string search is done in this case). An instruction of *Exc* instructs the test that the server's output is valid if it does not contain the specified value. In both cases, the content specification can include wild card patterns. For example, an Inc instruction can be *Inc:*Home page**. An Inc and an Exc instruction can be provided in quick succession in the following format: *Inc:*Home Page*,Exc:*home*. 12. **CREDENTIALS** – The **CREDENTIALS** parameter is to be set if a specific user name / password has to be specified to login to a page. Against this parameter, the **URLname** of every configured **URL** will be displayed; corresponding to each listed **URLname**, a **Username** text box and a **Password** text box will be made available. If the web server on which the test executes supports 'Anonymous user access', then this parameter will take either of the following values:

<table>
<tr><td></td><td colspan="3">

a valid Username and Password for every configured URLname
none in both the Username and Password text boxes of all configured URLnames (the default setting), if no user authorization is required

13. TIMEOUT - Here, specify the maximum duration (in seconds) for which the test will wait for a response from the server. The default TIMEOUT period is 30 seconds.
</td></tr>
<tr><td>Outputs of the test</td><td colspan="3">One set of results for every URL being monitored</td></tr>
<tr><td rowspan="2">Measurements of the test</td><td>Measurement</td><td>Measurement Unit</td><td>Interpretation</td></tr>
<tr><td>Availability:

This measurement indicates whether or not this URL* was successfully accessed and valid content was returned. In other words, this measure clearly indicates whether or not step (1) to step (12) of Figure 2.4 completed successfully and the client was able to view the complete contents of the URL, till the last byte of data.

*The term URL used in all discussions related to the measures of this test refers to the target URL indicated by step (1) of Figure 2.4, unless explicitly stated otherwise.</td><td>Percent</td><td>The value 100 for this measure indicates that the URL could be accessed successfully – i.e., the client could view the contents of the web page completely, till the last byte. The value 0 on the other hand indicates that the URL could not be accessed. This could be owing to a prolonged slowdown in the web server, performance issues with SiteMinder or an authentication failure reported by SiteMinder, processing bottlenecks experienced by the web agent, large size of the contents of the URL, etc. To zero-in on the exact reason for the inaccessibility of the configured URL, take a look at the values reported by the other measures of this test.

Note that even if the URL is not protected by SiteMinder, the 'Availability' measure will still report the value 100, provided the user does not receive any HTML response with a response code higher than 400. In this case, you can check the value reported by the 'URL SiteMinder protected' measure to figure out whether the URL is protected or not.</td></tr>
</table>

	Total response time: This measurement indicates the total time it took for the client to request for a URL and receive access to that URL via a session cookie. This is the sum total of the time taken to do the following: • The time taken by the client to know whether the URL is protected or not; • The time taken by the client to connect to the login URL and pass login credentials; • The time taken by the client to have his/her credentials authenticated by SitemInder; • The time taken by the client to view the complete contents of the requested URL, till its last byte; In short, this measure indicates the time taken to complete step **(1)** to step **(12)** of Figure 2.4 above.	Secs	Response time being high denotes a problem. Poor responsiveness can be caused due to a slowdown in the web server or SiteMinder, or because the content to be downloaded is large. Therefore, if the value of this measure is high or is increasing consistently, you will have to compare the values of the *Web server response time, URL redirect time*, *Siteminder authentication time*, and *Content download time* measures to figure out where the request spent maximum time; there is your bottleneck!

	Web server availability: Indicates whether or not the web server is able to process the request for this URL and return the location of the login URL to the user. In other words, this measure indicates whether or not step **(1)** to step **(4)** of Figure 2.4 could be completed successfully.	Percent	The value 100 for this measure indicates that the web server is available, has successfully determined the protection status of the URL, and has returned the location of the login URL to the user. If the measure reports the value 0, it is indicative of the non-availability of the web server. Availability is determined based on the response code returned by the server. If the server returns a response code between 200 and 300 the very first time the configured URL is hit, it indicates that the server is available. Any response code over 400 indicates non-availability of the web server. **Note that even if the URL is not protected by SiteMinder, the 'Web server availability' measure will still report the value 100, provided the user does not receive any HTML response with a response code higher than 400. In this case, you can check the value reported by the 'URL SiteMinder protected' measure to figure out whether the URL is protected or not.**
	Web server response time: Indicates the time taken to check and report whether this URL is protected or not; this is the time taken to perform steps **(1)**, **(2) and (3)** of Figure 2.4.	Secs	Response time being high denotes a problem. Poor response times may be due to the web server being overloaded or misconfigured. If the value of the *Total response time* measure is high, then compare the value of the *Web server response time* measure with the other response time measures reported by this test to determine whether a processing bottleneck in the web server is the reason why the client had trouble accessing the web site.
	URL redirect time: This measure indicates the time taken by the web agent on the web server to redirect the request to the login URL (indicated by step **(5)** of Figure 2.4); this is the total time taken to complete step **(5)** of Figure 2.4.	Secs	Ideally, the value of this measure should be low. A high value is indicative of poor responsiveness. If the value of the *Total response time* measure is high, then compare the value of the *URL redirect time* measure with the other response time measures reported by this test to determine whether a delay in sending the location of the login URL to the client is what caused accesses to the web site to slow down.

	URL SiteMinder protected: Indicates whether or not this URL is protected by SiteMinder; in other words, this measure checks whether the client received the login URL or not.	Boolean	While the value *1* for this measure indicates that the URL is protected by SiteMinder, the value 0 indicates that the URL is not protected.
	SiteMinder authentication status: Indicates whether user authentication succeeded or failed.	Percent	A value of 100% indicates that SiteMinder successfully authenticated the login credentials that were submitted to it by the web agent. The value 0 on the other hand indicates an authentication failure. If the client received an **SM_SESSION** cookie from the web agent, it indicates that authentication was successful. On the other hand, if the client did not receive the **SM_SESSION** cookie it is indicative of authentication failure. The probable cause for this failure is the submission of invalid/incorrect credentials to SiteMinder.
	SiteMinder authentication time: Indicates the time taken by SiteMinder to authenticate the user credentials; this is the time taken to perform steps **(6), (7)**, **(8)**, **and (9)** of Figure 2.4.	Secs	A low value is desired for this measure. A high value is indicative of an authentication bottleneck. If the value of the *Total response time* measure is high, then compare the value of the *SiteMinder authentication time* measure with the other response time measures reported by this test to determine whether a processing bottleneck in the SiteMinder Policy server is what is causing accesses to the web site to slow down.

	Content validity: This measure validates whether the content returned by this URL is valid or not.	Percent	A value of 100% indicates that the content returned by the test is valid. A value of 0% indicates that the content may not be valid. Content that matches the value of the **CONTENT** parameter of this test is deemed as valid content; in this case, the measure will report the value 100. If the actual content does not match the **CONTENT** specification, then the value of the measure will be 0. This capability for content validation is especially important for multi-tier web applications. For example, a user may not be able to login to the web site but the server may reply back with a valid HTML page where in the error message, say, "Invalid Login" is reported. In this case, the availability will be 100 % (since we got a valid HTML response). If the test is configured such that the content parameter should exclude the string "Invalid Login," in the above scenario content validity would have a value 0.
	Response code: Indicates the response code received by the client for the request to this URL.	Number	A value between 200 and 300 indicates a good response. A 4xx value indicates a problem with the requested content (eg., page not found). The value 401 is indicative of an authentication issue. A 5xx value indicates a server error.
	Content length: The size of the content returned by the server for the request to this URL.	Kbytes	Typically the content length returned by the server for a specific URL should be the same across time. Any change in this metric may indicate the need for further investigation on the server side.

	Content download time: Indicates the total time taken by the client to view the complete contents returned by this URL till the last byte of data; in other words, this is the time taken to complete steps **(11)** and **(12)** of Figure 2.4.	Secs	A high value for this measure denotes that the content is taking a long time to load. This could be owing to the presence of a large number of images in the content or one/more large images. You may want to check the value of the *Content length* measure to understand how large the content is. If the value of the *Total response time* measure is high, then compare the value of the *Content download time* measure with the other response time measures reported by this test to determine whether the size of the content poses a road-block to swift access to the protected web site.

2.3 The Web Site Layer

The **Web Site** layer is specific to each web site hosted on a web server. An internal WebSite test shown in Figure 2.8 provides a comprehensive view of the states of the individual web sites supported by a web server.

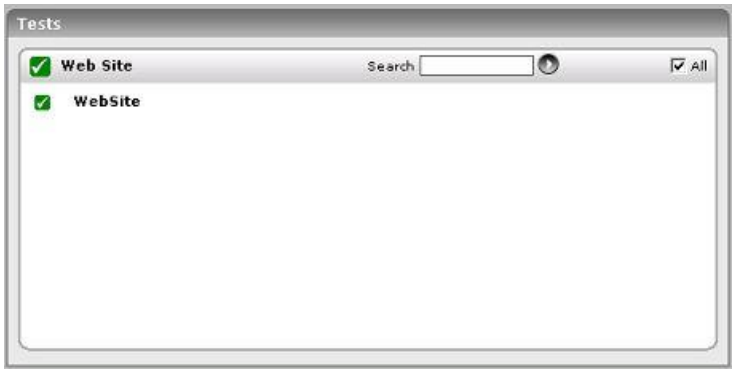


Figure 2.8: The WebSite test tracks the health of the Web Site layer

2.3.1 Web Site Test

Like the WebServer test, this test also interfaces with the eG web adapter to collect statistics relating to a web site.

Purpose	To measure the state of a web site (a single web server may support multiple web sites)
Target of the test	A web site supported by a web server
Agent deploying the test	An internal agent

Configurable parameters for the test	1. **TEST PERIOD** - How often should the test be executed 2. **HOST -** The host for which the test is to be configured 3. **PORT -** the port to which the specified **HOST** listens		
Outputs of the test	One set of results for every web site monitored		
Measurements made by the test	**Measurement**	**Measurement Unit**	**Interpretation**
	Connections: Rate of connections to the web site.	Conns/Sec	An increase or decrease in the connection rate can represent a change in the user workload.
	Requests: Rate of requests to the web site.	Reqs/Sec	With the advent of HTTP/1.1, multiple requests can be transmitted over the same TCP connection. The ratio of requests per connection can provide an idea of the effectiveness of the HTTP 1.1 protocol.
	Data transmitted: Rate at which the data is transmitted by the web site in response to user requests.	KB/Sec	A large increase in the data transmission rate can be indicative of an increase in the popularity of a web site hosted on the server.
	Data received: Rate at which the data is received by the web site.	KB/Sec	An increase in this value is indicative of an increase in user requests to the web site.
	Errors: Percentage of error responses from the web site.	Percent	Percentage of responses with a 400 or 500 status code.
	Aborts: Percentage of requests aborted by users (a request is deemed to have been aborted if either the connection is closed without any response being generated, or if the TCP connection is closed as the server is reading the request or as it is responding to the request).	Percent	A high percentage of aborts can happen if the web site's responsiveness has dramatically reduced. Aborts may also occur because of backend errors (e.g., application failures, database connection problems etc.).
	300 responses: Percentage of responses with a status code in the range 300-400.	Percent	Responses with a status code in the range 300-399 can indicate redirects from a server. Many a time, responses to cached content on the client also falls in this range.

	400 errors: Percentage of responses with a status code in the range 400-500.	Percent	An unexpected increase in the percentage of responses with status codes in the range 400-499 can indicate a sudden problem at the server.
	500 errors: Percentage of responses with a status code in the range 500-600.	Percent	Responses in the range 500-599 are caused by server-side errors (e.g., because of failures in the server side processing logic).
	Current requests: Number of server threads/processes currently in use for serving requests for a web site (this measurement is not available for Apache web servers).	Number	A majority of the server threads/processes being used simultaneously to serve requests for a web site may be indicative of a server bottleneck caused by the web site.

Note:

The following measures are not available in the Windows version of the product:

- Aborts
- 300 responses

2.4 The Web Transactions Layer

One of the unique capabilities of the eG Enterprise suite is its ability to monitor individual web transactions. The WebSiteTransaction test shown in Figure 2.9 is responsible for configuring an eG web adapter with the specifications of transactions corresponding to each web site being monitored. The web adapter is responsible for continuously tracking requests for these transactions and for periodically reporting a variety of statistics pertaining to each transaction back to the eG manager.

Figure 2.9: The tests that map to the Web Transactions layer of a web site.

2.4.1 Web Transactions Test

The Web Transactions test tracks the state of the individual transactions of a web site.

Purpose	To measure the state of individual transactions supported by a web site		
Target of the test	A web site supported by a web server		
Agent deploying the test	An internal agent		
Configurable parameters for the test	1. **TEST PERIOD** - How often should the test be executed 2. **HOST -** The host for which the test is to be configured 3. **PORT -** the port to which the specified host listens		
Outputs of the test	One set of results for each transaction supported by a web site		
Measurements made by the test	**Measurement**	**Measurement Unit**	**Interpretation**
	Requests: Rate of requests for a specific transaction.	Reqs/Sec	An increase or decrease in request rate for a specific transaction can represent a change in the user workload.
	Errors: Percentage of error-filled responses from the web site for a specific transaction.	Percent	Percentage of responses for the transaction that report a 400 or 500 status code.

	Aborts: Percentage of requests aborted by users when accessing a specific transaction from a web site (a request is deemed to have been aborted if either the connection is closed without any response being generated, or if the TCP connection is closed as the server is reading the request or as it is responding to the request).	Percent	A high percentage of aborts can happen if there has been a significant slow-down or error when users access a specific transaction (reasons for this could be errors in the application(s) supporting the transaction, failure of the backend database, etc.).
	Data transmitted: Rate at which the data is transmitted by the web site in response to user requests for a specific transaction.	KB/Sec	A large increase in the data transmission rate can be indicative of an increase in the popularity of a specific transaction. Alternatively, a sudden increase in data rate may also indicate a change in the characteristics of a transaction.
	Avg response time: The average time taken by the web site to respond to requests for a specific transaction, measured in seconds. Only requests for which successful responses are received are considered while computing the average response time.	Secs	An increase in response time is a major cause for user dissatisfaction with e-business sites. By correlating an increase in response time with the other metrics collected by the agent per transaction, an operator can diagnose the reason(s) for the deterioration in performance.
	Current requests: Number of server threads/processes currently in use for serving requests for the specific transaction supported by a web site (this measurement is not available for Apache web servers).	Number	If a majority of the server threads/processes are in use simultaneously to serve requests for a specific transaction, this may be indicative of a server bottleneck caused by the transaction being considered.

Note:

- For the Web Transactions test to run smoothly, logging will have to be enabled for the web site(s) being monitored. To enable logging, follow the procedure discussed in the *eG Installation Guide*.
- The *Aborts* and *Current requests* measures will not be available for Windows versions of the product.
- The Apache web server handles requests by spawning multiple threads/processes. As each thread/process handles requests, the eG instrumentation in the thread/process collects statistics for each request. Periodically, a summary of the new requests handled by a thread/process is updated to a common shared memory location from which the eG agent collects metrics and reports them to the eG manager. The frequency at which a thread/process updates the common shared memory is governed by an environment variable REQUEST_INTERVAL. By default, if this variable is not set, the eG web adapter uses a value of 5. That is, each thread/process updates the shared memory location once for every 5 requests that it handles. In a production system, with real user activity, this default setting will ensure that the overheads of the web adapter are minimal. In testing/staging environments, if the load on the web server is very low, the default **REQUEST_INTERVAL** setting may mean that the threads/processes do not update the shared memory location frequently. Hence, the administrator may notice slight discrepancies between the load imposed on the web server and the metrics reported by the web adapter. In such situations, the administrator can set the **REQUEST_INTERVAL** environment variable to 1 in the Apache server's start up script to force the Apache threads/processes to update the shared memory upon processing each request.

2.5 Customizing Applications for Monitoring by eG Enterprise's Web Adapter

The HTTP protocol specification provides various status codes that are used by web applications to indicate error conditions. For example, a response code of 404 indicates that a specific page was not found on the server. Likewise, a response code of 500 indicates a server-side processing error. As the Web has evolved to support a variety of complex applications that involve dynamic rather than static content, application developers have resorted to newer methods of informing users of application-level problems. For example, rather than returning a 404 response code to indicate missing content (which results in the browser throwing an error message), an application developer may choose to report the error by formatting it within a HTML page (i.e., by providing a 200 response code) and providing additional user-friendly error messages such as the email address of the web site's administrator. The side effect of this approach is that the large number of existing monitoring tools that primarily use the HTTP response code to detect application failures will not be able to effectively detect and report problem conditions.

To enable eG agents to detect and report on such application-specific problems, eG Enterprise's web adapter allows applications to use a specific HTTP header variable called **`Eg-Status`** to report error conditions to it. To report a specific problem, application developers should assign corresponding error codes (following the HTTP protocol specification's response code convention). For example, to report an application-specific error, an application that

uses Java technologies (JSP, Servlets, etc.) can incorporate the following code to set the Eg-Status header value while generating a HTTP response:

```
response.setHeader("Eg-Status", "500");
```

The eG web adapter searches the HTTP header of all responses generated by a web server. If the Eg-Status header exists, the value of this variable is used to override the HTTP response code value. This method allows application developers to indicate potential error conditions to the eG agents, without affecting the output being provided by their applications to users.

2.6 Troubleshooting Apache/IBM HTTP/iPlanet Web Server Monitoring

If the *Web Site* and *Web Transactions* tests fail for a web server that is running on a Unix platform, then, check whether the eG web adapter has been properly configured on Unix. If the web adapter has been configured properly, then check whether the web server being monitored is a 32-bit or a 64-bit application. Then, check whether the eG agent installed on the web server is a 32-bit or a 64-bit agent.

If the bit-rates of the eG agent and the web server do not match, the *Web Site* test and the *Web Transactions* test will fail. This is because, a 32-bit web server on Unix can only be monitored by a 32-bit eG agent; likewise, a 64-bit web server on Unix, can only be monitored by a 64-bit eG agent. In the event of a mismatch therefore, uninstall the eG agent that pre-exists and install an eG agent with the same bit-rate as the web server.

Chapter 3

Monitoring IIS Web Servers

Internet Information Services (IIS) is a powerful Web server that provides a highly reliable, manageable, and scalable Web application infrastructure for Windows servers.

eG Enterprise offers a specialized *IIS* web server model for monitoring an IIS web server. Figure 3.1 depicts the model used by the eG Enterprise suite to monitor an Internet Information Server.

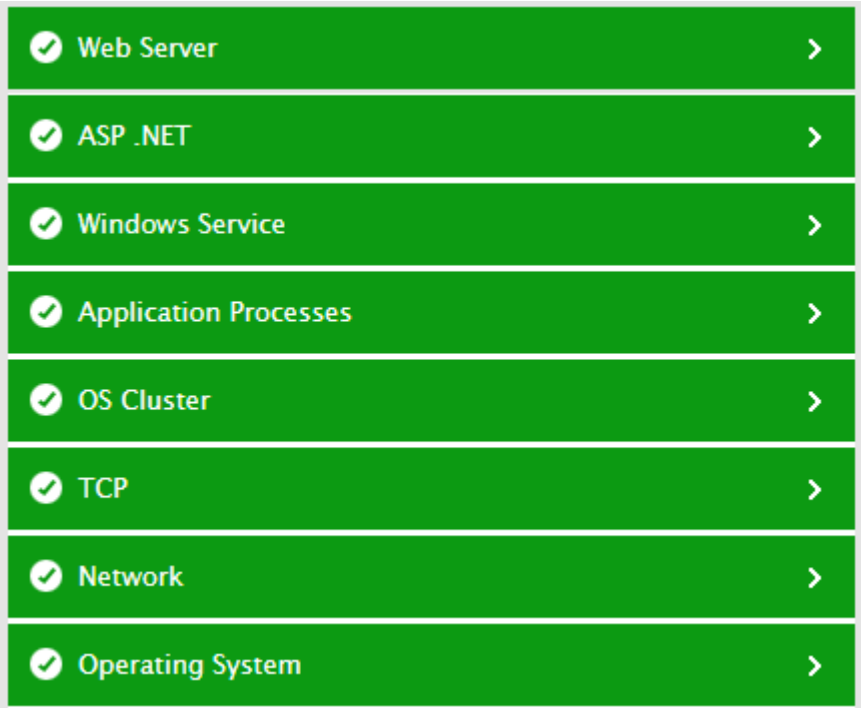


Figure 3.1: The different layers that the eG Enterprise suite monitors for an IIS web server

The **Operating System**, **Network**, and **Tcp** layers of an *IIS* web server model are similar to that of a *Windows Generic* server model. Since these tests have been dealt with in the *Monitoring Unix and Windows Servers* document, Section 3.1 focuses on the **Application Processes** layer.

Similarly, the **Web Site** and **Web Transactions** layers have also been described in detail in Chapter 2. Section 3.2 therefore, discusses the **Web Server** layer of the IIS web server.

Note:

- The eG agent is capable of monitoring IIS web servers (ver. 4, 5, 6, and 7) in an agent-based and an agentless manner; however, note that, in the agentless mode, the solution cannot monitor web transactions to web sites configured on the target IIS web server.
- To monitor an IIS web server on Windows 2008, you need to configure a **Web Server** role on the target Windows 2008 server. To know how to configure the **Web Server** role, refer to the *eG Installation Guide*.

3.1 The Application Processes Layer

The **Application Processes** layer of an IIS web server periodically checks whether the critical web server processes are running or not, and also reports the percentage of CPU and memory resources utilized by these processes. In addition, the layer also monitors the eventlogs on the IIS web server host from time to time, so that critical application and system-related errors are instantly detected.

Figure 3.2: The tests mapped to the Application Processes layer

For details on the **Processes** test and the **WindowsProcesses** test in Figure 3.2, please refer to the *Monitoring Unix and Windows Servers* document. The sections below discuss the **ApplicationEvents** and **SystemEvents** tests only.

3.1.1 Application Events Test

This test reports the statistical information about the application events generated by the target system.

Configurable parameters for the test	1. **TEST PERIOD** - How often should the test be executed 2. **HOST** - The host for which the test is to be configured 3. **PORT –** Refers to the port used by the EventLog Service. Here it is null. 4. **LOGTYPE** – Refers to the type of event logs to be monitored. The default value is *application*. 5. **POLICY BASED FILTER** - Using this page, administrators can configure the event sources, event IDs, and event descriptions to be monitored by this test. In order to enable administrators to easily and accurately provide this specification, this page provides the following options: ➢ Manually specify the event sources, IDs, and descriptions in the **FILTER** text area, or, ➢ Select a specification from the predefined filter policies listed in the **FILTER** box For explicit, manual specification of the filter conditions, select the **NO** option against the **POLICY BASED FILTER** field. This is the default selection. To choose from the list of pre-configured filter policies, or to create a new filter policy and then associate the same with the test, select the **YES** option against the **POLICY BASED FILTER** field. 6. **FILTER** - If the **POLICY BASED FILTER** flag is set to **NO**, then a **FILTER** text area will appear, wherein you will have to specify the event sources, event IDs, and event descriptions to be monitored. This specification should be of the following format: *{Displayname}:{event_sources_to_be_included}:{event_sources_to_be_excluded}:{event_IDs_to_be_included}:{event_IDs_to_be_excluded}:{event_descriptions_to_be_included}:{event_descriptions_to_be_excluded}*. For example, assume that the **FILTER** text area takes the value, *OS_events:all:Browse,Print:all:none:all:none*. Here: c. *OS_events* is the display name that will appear as a descriptor of the test in the monitor UI; d. *all* indicates that all the event sources need to be considered while monitoring. To monitor specific event sources, provide the source names as a comma-separated list. To ensure that none of the event sources are monitored, specify *none*. e. Next, to ensure that specific event sources are excluded from monitoring, provide a comma-separated list of source names. Accordingly, in our example, *Browse* and *Print* have been excluded from monitoring. Alternatively, you can use *all* to indicate that all the event sources have to be excluded from monitoring, or *none* to denote that none of the event sources need be excluded. f. In the same manner, you can provide a comma-separated list of event IDs that require monitoring. The *all* in our example represents that all the event IDs need to be considered while monitoring.

- Similarly, the *none* (following *all* in our example) is indicative of the fact that none of the event IDs need to be excluded from monitoring. On the other hand, if you want to instruct the eG Enterprise system to ignore a few event IDs during monitoring, then provide the IDs as a comma-separated list. Likewise, specifying *all* makes sure that all the event IDs are excluded from monitoring.
- The *all* which follows implies that all events, regardless of description, need to be included for monitoring. To exclude all events, use *none*. On the other hand, if you provide a comma-separated list of event descriptions, then the events with the specified descriptions will alone be monitored. Event descriptions can be of any of the following forms - *desc**, or *desc*, or **desc**,or *desc**, or *desc1*desc2*, etc. *desc* here refers to any string that forms part of the description. A leading '*' signifies any number of leading characters, while a trailing '*' signifies any number of trailing characters.
- In the same way, you can also provide a comma-separated list of event descriptions to be excluded from monitoring. Here again, the specification can be of any of the following forms: *desc**, or *desc*, or **desc**,or *desc**, or *desc1*desc2*, etc. *desc* here refers to any string that forms part of the description. A leading '*' signifies any number of leading characters, while a trailing '*' signifies any number of trailing characters. In our example however, none is specified, indicating that no event descriptions are to be excluded from monitoring. If you use *all* instead, it would mean that all event descriptions are to be excluded from monitoring.

By default, the **FILTER** parameter contains the value*: all:all:none:all:none:all:none*. Multiple filters are to be separated by semi-colons (;).

Note:

The event sources and event IDs specified here should be exactly the same as that which appears in the Event Viewer window.

On the other hand, if the **POLICY BASED FILTER** flag is set to **YES**, then a **FILTER** list box will appear, displaying the filter policies that pre-exist in the eG Enterprise system. A filter policy typically comprises of a specific set of event sources, event IDs, and event descriptions to be monitored. This specification is built into the policy in the following format:

{Policyname}:{event_sources_to_be_included}:{event_sources_to_be_excluded}:{event_IDs_to_be_included}:{event_IDs_to_be_excluded}:{event_descriptions_to_be_included}:{event_descriptions_to_be_excluded}

To monitor a specific combination of event sources, event IDs, and event descriptions, you can choose the corresponding filter policy from the **FILTER** list box. Multiple filter policies can be so selected. Alternatively, you can modify any of the existing policies to suit your needs, or create a new filter policy. To facilitate this, a **Click here** link appears just above the test configuration section, once the **YES** option is chosen against **POLICY BASED FILTER**. Clicking on the **Click here** link leads you to a page where you can modify the existing policies or create a new one (refer to page 39). The changed policy or the new policy can then be associated with the test by selecting the policy name from the **FILTER** list box in this page.

	7. **USEWMI** - The eG agent can either use WMI to extract event log statistics or directly parse the event logs using event log APIs. If the **USEWMI** flag is **YES**, then WMI is used. If not, the event log APIs are used. This option is provided because on some Windows 2000 systems (especially ones with service pack 3 or lower), the use of WMI access to event logs can cause the CPU usage of the WinMgmt process to shoot up. On such systems, set the **USEWMI** parameter value to **NO**. 8. **STATELESS ALERTS** - Typically, the eG manager generates email alerts only when the state of a specific measurement changes. A state change typically occurs only when the threshold of a measure is violated a configured number of times within a specified time window. While this ensured that the eG manager raised alarms only when the problem was severe enough, in some cases, it may cause one/more problems to go unnoticed, just because they did not result in a state change. For example, take the case of the EventLog test. When this test captures an error event for the very first time, the eG manager will send out a **CRITICAL** email alert with the details of the error event to configured recipients. Now, the next time the test runs, if a different error event is captured, the eG manager will keep the state of the measure as **CRITICAL**, but will not send out the details of this error event to the user; thus, the second issue will remain hidden from the user. To make sure that administrators do not miss/overlook critical issues, the eG Enterprise monitoring solution provides the **stateless alerting** capability. To enable this capability for this test, set the **STATELESS ALERTS** flag to **Yes**. This will ensure that email alerts are generated for this test, regardless of whether or not the state of the measures reported by this test changes. 9. **EVENTS DURING RESTART** - By default, the **EVENTS DURING RESTART** flag is set to **Yes**. This ensures that whenever the agent is stopped and later started, the events that might have occurred during the period of non-availability of the agent are included in the number of events reported by the agent. Setting the flag to **No** ensures that the agent, when restarted, ignores the events that occurred during the time it was not available. 10. **DDFORINFORMATION** – eG Enterprise also provides you with options to restrict the amount of storage required for event log tests. Towards this end, the **DDFORINFORMATION** and **DDFORWARNING** flags have been made available in this page. By default, both these flags are set to **Yes**, indicating that by default, the test generates detailed diagnostic measures for information events and warning events. If you do not want the test to generate and store detailed measures for information events, set the **DDFORINFORMATION** flag to **No**. 11. **DDFORWARNING** – To ensure that the test does not generate and store detailed measures for warning events, set the **DDFORWARNING** flag to **No**. 12. **DD FREQUENCY** - Refers to the frequency with which detailed diagnosis measures are to be generated for this test. The default is *1:1*. This indicates that, by default, detailed measures will be generated every time this test runs, and also every time the test detects a problem. You can modify this frequency, if you so desire. Also, if you intend to disable the detailed diagnosis capability for this test, you can do so by specifying *none* against **DD FREQUENCY**.

	13. **DETAILED DIAGNOSIS** - To make diagnosis more efficient and accurate, the eG Enterprise suite embeds an optional detailed diagnostic capability. With this capability, the eG agents can be configured to run detailed, more elaborate tests as and when specific problems are detected. To enable the detailed diagnosis capability of this test for a particular server, choose the **On** option. To disable the capability, click on the **Off** option. The option to selectively enabled/disable the detailed diagnosis capability will be available only if the following conditions are fulfilled: ➢ The eG manager license should allow the detailed diagnosis capability ➢ Both the normal and abnormal frequencies configured for the detailed diagnosis measures should not be 0.		
Outputs of the test	One set of results for the **FILTER** configured		
Measurements made by the test	**Measurement**	**Measurement Unit**	**Interpretation**
	Application errors: This refers to the number of application error events that were generated.	Number	A very low value (zero) indicates that the system is in a healthy state and all applications are running smoothly without any potential problems. An increasing trend or high value indicates the existence of problems like loss of functionality or data in one or more applications. Please check the Application Logs in the Event Log Viewer for more details.
	Application information count: This refers to the number of application information events generated when the test was last executed.	Number	A change in the value of this measure may indicate infrequent but successful operations performed by one or more applications. Please check the Application Logs in the Event Log Viewer for more details.
	Application warnings: This refers to the number of warnings that were generated when the test was last executed.	Number	A high value of this measure indicates application problems that may not have an immediate impact, but may cause future problems in one or more applications. Please check the Application Logs in the Event Log Viewer for more details.

a. The filter policy for the ApplicationEvents test and SystemEvents test typically comprises of a specific set of event sources, event IDs, and event descriptions to be monitored. This specification is expressed by the eG Enterprise system in the following format:

b. *{Policyname}:{event_sources_to_be_included}:{event_sources_to_be_excluded}:{event_IDs_to_be_included}:{event_IDs_to_be_excluded}:{event_descriptions_to_be_included}:{event_descriptions_to_be_excluded}*

To add a new policy, do the following:

1. Click on the **Click here** hyperlink available just above the test configuration of the ApplicationEvents test or SystemEvents test (see Figure 3.3).

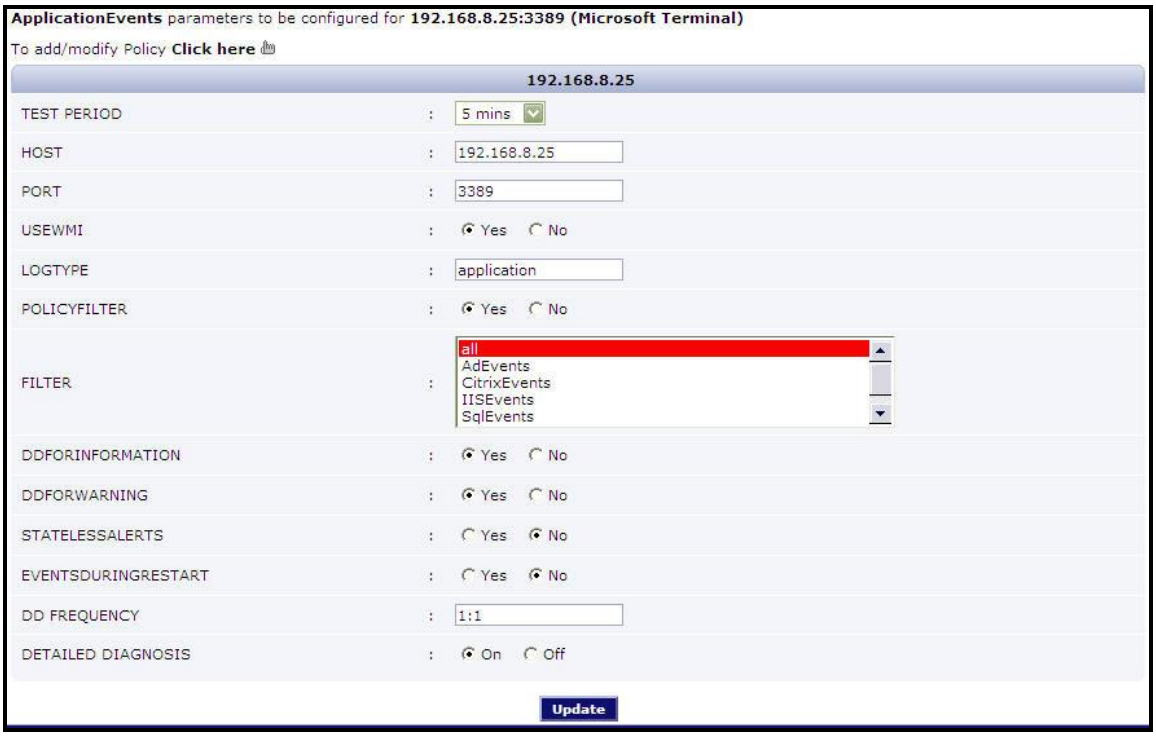


Figure 3.3: Configuring an ApplicationEvents test

2. Figure 3.4 will then appear listing the policies that pre-exist.

Figure 3.4: List of policies

3. To view the contents of a policy, click on the **View** button against the policy name. While a policy can be modified by clicking on the **Modify** button, it can be deleted using the **Delete** button. The default policy is **all**, which can only be viewed and **not modified** or **deleted**. The specification contained within this policy is: *all:none:all:none:all:none*.

4. To create a new policy, click on the **Add New Policy** button in Figure 3.8. Doing so invokes Figure 3.5, using which a new policy can be created.

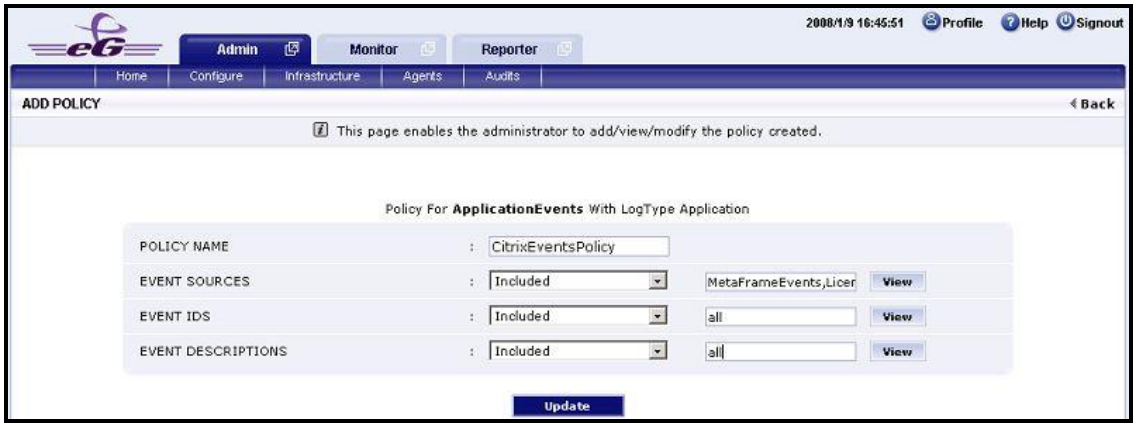


Figure 3.5: Adding a new filter policy

5. In Figure 3.5, first, provide a unique name against **POLICY NAME**.

6. To include one/more event sources for monitoring, select **Included** from the **EVENT SOURCES** drop-down list, and then specify a comma-separated list of event sources in the adjacent text box. If you require more space to specify the event sources, click on the **View** button next to the text box. This will invoke an **EVENT SOURCES INCLUDED** text area (see Figure 3.6), wherein the specification can be provided more clearly and comfortably.

Figure 3.6: Viewing the text area

7. To exclude specific event sources from monitoring, select **Excluded** from the **EVENT SOURCES** drop-down list, and then specify a comma-separated list of event sources to be excluded in the adjacent text box. If you require more space to specify the event sources, click on the **View** button next to the text box. This will invoke an **EVENT SOURCES EXCLUDED** text area, wherein the specification can be provided more clearly and comfortably.

Note:

At any given point in time, you can choose to either **Include** or **Exclude** event sources, but you cannot do both. If you have chosen to include event sources, then the eG Enterprise system automatically assumes that no event sources need be excluded. Accordingly, the *{event_sources_to_be_excluded}* section of the filter format mentioned above, will assume the value *none*. Similarly, if you have chosen to exclude specific event sources from monitoring, then the *{event_sources_to_be_included}* section of the format above will automatically take the value *all*, indicating that all event sources except the ones explicitly excluded, will be included for monitoring.

8. In the same way, select **Included** from the **EVENT IDS** list and then, provide a comma-separated list of event IDs to be monitored. For more space, click on the **View** button next to the text box, so that an **EVENT IDS INCLUDED** text area appears.
9. If you, on the other hand, want to exclude specific event IDs from monitoring, then first select **Excluded** from the **EVENT IDS** list box, and then provide a comma-separated list of event IDs to be excluded. For more space, click on the **View** button next to the text box, so that an **EVENT IDS EXCLUDED** text area appears.

Note:

At any given point in time, you can choose to either **Include** or **Exclude** event IDs, but you cannot do both. If you have chosen to include event IDs, then the eG Enterprise system automatically assumes that no event IDs need be excluded. Accordingly, the *{event_IDs_to_be_excluded}* section of the filter format mentioned above, will assume the value *none*. Similarly, if you have chosen to exclude specific event IDs from monitoring, then the *{event_IDs_to_be_included}* section of the format above will automatically take the value *all*, indicating that all event IDs except the ones explicitly excluded, will be included for monitoring.

10. Likewise, select **Included** from the **EVENT DESCRIPTIONS** list and then, provide a comma-separated list of event descriptions to be monitored. For more space, click on the **View** button next to the text box, so that an **EVENT DESCRIPTIONS INCLUDED** text area appears.
11. For excluding specific event descriptions from monitoring, first select **Excluded** from the **EVENT DESCRIPTIONS** list box, and then provide a comma-separated list of event descriptions to be excluded. For more space, click on the **View** button next to the text box, so that an **EVENT DESCRIPTIONS EXCLUDED** text area appears.

Note:

At any given point in time, you can choose to either **Include** or **Exclude** event descriptions, but you cannot do both. If you have chosen to include event descriptions, then the eG Enterprise system automatically assumes that no event descriptions need be excluded. Accordingly, the *{event_descriptions_to_be_excluded}* section of the filter format mentioned above, will assume the value *none*. Similarly, if you have chosen to exclude specific event descriptions from monitoring, then the *{event_descriptions_to_be_included}* section of the format above will automatically take the value *all*. This indicates that all event descriptions except the ones explicitly excluded, will be included for monitoring.

12. Finally, click the **Update** button.
13. The results of the configuration will then be displayed as depicted by Figure 3.7

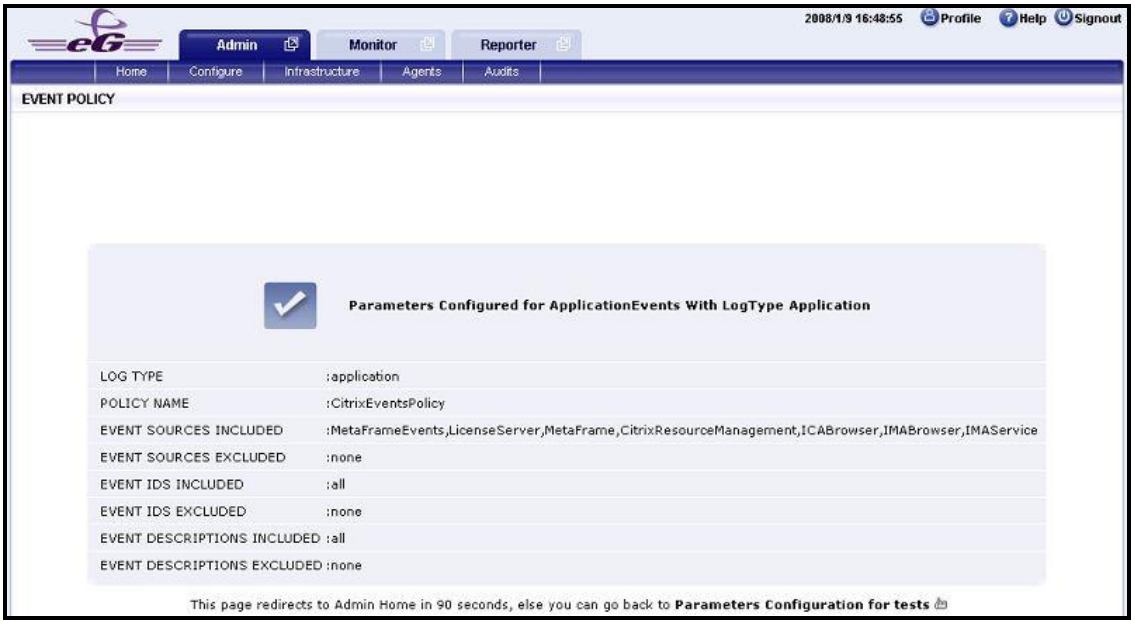


Figure 3.7: Results of the configuration

3.1.2 System Events Test

This test reports the statistical information about the system events generated by the target system.

Configurable parameters for the test	1. **TEST PERIOD** - How often should the test be executed 2. **HOST** - The host for which the test is to be configured 3. **PORT –** Refers to the port used by the EventLog Service. Here it is null. 4. **LOGTYPE** – Refers to the type of event logs to be monitored. The default value is *application*. 5. **POLICY BASED FILTER** - Using this page, administrators can configure the event sources, event IDs, and event descriptions to be monitored by this test. In order to enable administrators to easily and accurately provide this specification, this page provides the following options: ➢ Manually specify the event sources, IDs, and descriptions in the **FILTER** text area, or, ➢ Select a specification from the predefined filter policies listed in the **FILTER** box For explicit, manual specification of the filter conditions, select the **NO** option against the **POLICY BASED FILTER** field. This is the default selection. To choose from the list of pre-configured filter policies, or to create a new filter policy and then associate the same with the test, select the **YES** option against the **POLICY BASED FILTER** field. 6. **FILTER** - If the **POLICY BASED FILTER** flag is set to **NO**, then a **FILTER** text area will appear, wherein you will have to specify the event sources, event IDs, and event descriptions to be monitored. This specification should be of the following format: *{Displayname}:{event_sources_to_be_included}:{event_sources_to_be_excluded}:{event_IDs_to_be_included}:{event_IDs_to_be_excluded}:{event_descriptions_to_be_included}:{event_descriptions_to_be_excluded}*. For example, assume that the **FILTER** text area takes the value, *OS_events:all:Browse,Print:all:none:all:none*. Here: g. *OS_events* is the display name that will appear as a descriptor of the test in the monitor UI; h. *all* indicates that all the event sources need to be considered while monitoring. To monitor specific event sources, provide the source names as a comma-separated list. To ensure that none of the event sources are monitored, specify *none*. i. Next, to ensure that specific event sources are excluded from monitoring, provide a comma-separated list of source names. Accordingly, in our example, *Browse* and *Print* have been excluded from monitoring. Alternatively, you can use *all* to indicate that all the event sources have to be excluded from monitoring, or *none* to denote that none of the event sources need be excluded. j. In the same manner, you can provide a comma-separated list of event IDs that require monitoring. The *all* in our example represents that all the event IDs need to be considered while monitoring.

- Similarly, the *none* (following *all* in our example) is indicative of the fact that none of the event IDs need to be excluded from monitoring. On the other hand, if you want to instruct the eG Enterprise system to ignore a few event IDs during monitoring, then provide the IDs as a comma-separated list. Likewise, specifying *all* makes sure that all the event IDs are excluded from monitoring.
- The *all* which follows implies that all events, regardless of description, need to be included for monitoring. To exclude all events, use *none*. On the other hand, if you provide a comma-separated list of event descriptions, then the events with the specified descriptions will alone be monitored. Event descriptions can be of any of the following forms - *desc**, or *desc*, or **desc**,or *desc**, or *desc1*desc2*, etc. *desc* here refers to any string that forms part of the description. A leading '*' signifies any number of leading characters, while a trailing '*' signifies any number of trailing characters.
- In the same way, you can also provide a comma-separated list of event descriptions to be excluded from monitoring. Here again, the specification can be of any of the following forms: *desc**, or *desc*, or **desc**,or *desc**, or *desc1*desc2*, etc. *desc* here refers to any string that forms part of the description. A leading '*' signifies any number of leading characters, while a trailing '*' signifies any number of trailing characters. In our example however, none is specified, indicating that no event descriptions are to be excluded from monitoring. If you use *all* instead, it would mean that all event descriptions are to be excluded from monitoring.

By default, the **FILTER** parameter contains the value*: all:all:none:all:none:all:none*. Multiple filters are to be separated by semi-colons (;).

Note:

The event sources and event IDs specified here should be exactly the same as that which appears in the Event Viewer window.

On the other hand, if the **POLICY BASED FILTER** flag is set to **YES**, then a **FILTER** list box will appear, displaying the filter policies that pre-exist in the eG Enterprise system. A filter policy typically comprises of a specific set of event sources, event IDs, and event descriptions to be monitored. This specification is built into the policy in the following format:

{Policyname}:{event_sources_to_be_included}:{event_sources_to_be_excluded}:{event_IDs_to_be_included}:{event_IDs_to_be_excluded}:{event_descriptions_to_be_included}:{event_descriptions_to_be_excluded}

To monitor a specific combination of event sources, event IDs, and event descriptions, you can choose the corresponding filter policy from the **FILTER** list box. Multiple filter policies can be so selected. Alternatively, you can modify any of the existing policies to suit your needs, or create a new filter policy. To facilitate this, a **Click here** link appears just above the test configuration section, once the **YES** option is chosen against **POLICY BASED FILTER**. Clicking on the **Click here** link leads you to a page where you can modify the existing policies or create a new one (refer to page 39). The changed policy or the new policy can then be associated with the test by selecting the policy name from the **FILTER** list box in this page.

	7. **USEWMI** - The eG agent can either use WMI to extract event log statistics or directly parse the event logs using event log APIs. If the **USEWMI** flag is **YES**, then WMI is used. If not, the event log APIs are used. This option is provided because on some Windows 2000 systems (especially ones with service pack 3 or lower), the use of WMI access to event logs can cause the CPU usage of the WinMgmt process to shoot up. On such systems, set the **USEWMI** parameter value to **NO**. 8. **STATELESS ALERTS** - Typically, the eG manager generates email alerts only when the state of a specific measurement changes. A state change typically occurs only when the threshold of a measure is violated a configured number of times within a specified time window. While this ensured that the eG manager raised alarms only when the problem was severe enough, in some cases, it may cause one/more problems to go unnoticed, just because they did not result in a state change. For example, take the case of the EventLog test. When this test captures an error event for the very first time, the eG manager will send out a **CRITICAL** email alert with the details of the error event to configured recipients. Now, the next time the test runs, if a different error event is captured, the eG manager will keep the state of the measure as **CRITICAL**, but will not send out the details of this error event to the user; thus, the second issue will remain hidden from the user. To make sure that administrators do not miss/overlook critical issues, the eG Enterprise monitoring solution provides the **stateless alerting** capability. To enable this capability for this test, set the **STATELESS ALERTS** flag to **Yes**. This will ensure that email alerts are generated for this test, regardless of whether or not the state of the measures reported by this test changes. 9. **EVENTS DURING RESTART** - By default, the **EVENTS DURING RESTART** flag is set to **Yes**. This ensures that whenever the agent is stopped and later started, the events that might have occurred during the period of non-availability of the agent are included in the number of events reported by the agent. Setting the flag to **No** ensures that the agent, when restarted, ignores the events that occurred during the time it was not available. 10. **DDFORINFORMATION** – eG Enterprise also provides you with options to restrict the amount of storage required for event log tests. Towards this end, the **DDFORINFORMATION** and **DDFORWARNING** flags have been made available in this page. By default, both these flags are set to **Yes**, indicating that by default, the test generates detailed diagnostic measures for information events and warning events. If you do not want the test to generate and store detailed measures for information events, set the **DDFORINFORMATION** flag to **No**. 11. **DDFORWARNING** – To ensure that the test does not generate and store detailed measures for warning events, set the **DDFORWARNING** flag to **No**. 12. **DD FREQUENCY** - Refers to the frequency with which detailed diagnosis measures are to be generated for this test. The default is *1:1*. This indicates that, by default, detailed measures will be generated every time this test runs, and also every time the test detects a problem. You can modify this frequency, if you so desire. Also, if you intend to disable the detailed diagnosis capability for this test, you can do so by specifying *none* against **DD FREQUENCY**.

	13. **DETAILED DIAGNOSIS** - To make diagnosis more efficient and accurate, the eG Enterprise suite embeds an optional detailed diagnostic capability. With this capability, the eG agents can be configured to run detailed, more elaborate tests as and when specific problems are detected. To enable the detailed diagnosis capability of this test for a particular server, choose the **On** option. To disable the capability, click on the **Off** option. The option to selectively enabled/disable the detailed diagnosis capability will be available only if the following conditions are fulfilled: • The eG manager license should allow the detailed diagnosis capability • Both the normal and abnormal frequencies configured for the detailed diagnosis measures should not be 0.		
Outputs of the test	One set of results for the **FILTER** configured		
Measurements made by the test	**Measurement**	**Measurement Unit**	**Interpretation**
	System errors: This refers to the number of system error events generated during the last execution of the test.	Number	A very low value (zero) indicates that the system is in healthy state and all Windows services and low level drivers are running without any potential problems. An increasing trend or a high value indicates the existence of problems such as loss of functionality or data in one or more Windows services and low level drivers. Please check the Application Logs in the Event Log Viewer for more details.
	System information messages: This refers to the number of service-related and driver-related information events that were generated during the test's last execution.	Number	A change in value of this measure may indicate infrequent but successful operations performed by one or more applications. Please check the Application Logs in the Event Log Viewer for more details.
	System warnings: This refers to the number of service-related and driver-related warnings generated in the during the test's last execution.	Number	A high value of this measure indicates problems that may not have an immediate impact, but may cause future problems in one or more Windows servers and low level drivers. Please check the Application Logs in the Event Log Viewer for more details.

Note:

The **STATELESS ALERTING** capability is currently available for the following tests alone, by default:

- EventLog test
- ApplicationEventLog test
- SystemEventLog test
- ApplicationEvents test
- SystemEvents test
- SecurityLog test

If need be, you can enable the **stateless alerting** capability for other tests. To achieve this, follow the steps given below:

- Login to the eG manager host.
- Edit the **eg_specs.ini** file in the **<EG_INSTALL_DIR>\manager\config** directory.
- Locate the test for which the **Stateless Alarms** flag has to be enabled.
- Insert the entry, **-statelessAlerts yes**, into the test specification as depicted below:

```
EventLogTest::$hostName:$portNo=$hostName, -auto, -host $hostName -port
$portNo -eventhost $hostIp -eventsrc all -excludedSrc none -useWmi yes -
statelessAlerts yes -ddFreq 1:1 -rptName $hostName, 300
```

Once the **stateless alerting capability** is enabled for a test (as discussed above), you will find that everytime the test reports a problem, the eG manager does the following:

- Closes the alarm that pre-exists for that problem;
- Sends out a normal alert indicating the closure of the old problem;
- Opens a new alarm and assigns a new alarm ID to it;
- Sends out a fresh email alert to the configured users, intimating them of the new issue.

In a redundant manager setup, the secondary manager automatically downloads the updated **eg_specs.ini** file from the primary manager, and determines whether the stateless alerting capability has been enabled for any of the tests reporting metrics to it. If so, everytime a threshold violation is detected by such a test, the secondary manager will perform the tasks discussed above for the problem reported by that test. Similarly, the primary manager will check whether the stateless alert flag has been switched on for any of the tests reporting to it, and if so, will automatically perform the above-mentioned tasks whenever those tests report a deviation from the

Note:

- Since alerts will be closed after every measurement period, alarm escalation will no longer be relevant for tests that have **statelessAlerts** set to **yes**.
- For tests with **statelessAlerts** set to **yes**, **statelessAlerts** will apply for all measurements of that test (i.e., it will not be possible to only have one of the measurements with stateless alerts and others without).
- If **statelessAlerts** is set to **yes** for a test, an alarm will be opened during one measurement period (if a threshold violation happens) and will be closed prior to the next measurement period. This way, if a threshold violation happens in successive measurement periods, there will be one alarm per measurement period. This will reflect in all the corresponding places in the eG Enterprise system. For example, multiple alerts in successive measurement periods will result in multiple trouble tickets being opened (one for each measurement period). Likewise, the alarm history will also show alarms being opened during a measurement period and closed during the next measurement period.

3.2 The ASP .NET Layer

The tests mapped to the ASP .NET layer report on the health of the CLR (Common Language Runtime) of the ASP .NET framework that the IIS web server supports, and that of the AppDomains in which the .NET applications run.

3.2.1 ASP.Net SQL Data Provider Test

A data provider in the .NET Framework serves as a bridge between an application and a data source. A .NET Framework data provider enables you to return query results from a data source, execute commands at a data source, and propagate changes in a DataSet to a data source.

The .Net Data Provider for SQL Server allows you to connect to Microsoft SQL Server 7.0, 2000, and 2005 databases, and perform the above-mentioned operations. This test reports many useful metrics that shed light on the health of the interactions between the ASP .Net sever and the SQL server.

Purpose	Shed light on the health of the interactions between the ASP .Net sever and the SQL server
Target of the test	The ASP .Net server
Agent deploying the test	An internal agent
Configurable parameters for the test	1. **TEST PERIOD** - How often should the test be executed 2. **HOST** - The host for which the test is to be configured 3. **PORT** - The port at which the specified host listens

Outputs of the test	One set of results for the ASP .Net server being monitored		
Measurements made by the test	**Measurement**	**Measurement Unit**	**Interpretation**
	Hard connects: Indicates the number of actual connections per second that are being made to a database server.	Connects/Sec	
	Hard disconnects: Indicates the number of actual disconnects per second that are being made to a database server.	Disconnects/Sec	
	Active connection pool groups: Indicates the number of currently active connection pool groups.	Number	The value of this measure is controlled by the number of unique connection strings that are found in the AppDomain.

	Active connection pools: Indicates the number of connection pools that are currently active.	Number	When a connection is first opened, a connection pool is created based on matching criteria that associates the pool with the connection string in the connection. Each connection pool is associated with a distinct connection string. If the connection string is not an exact match to an existing pool when a new connection is opened, a new pool is created. Connections are pooled per process, per application domain, per connection string, and, when integrated security is used, per Windows identity. When using Windows Authentication (integrated security), both the *Active connection pool groups* and *Active connection pools* measures are significant. The reason is that connection pool groups map to unique connection strings. When integrated security is used, connection pools map to connection strings and additionally create separate pools for individual Windows identities. For example, if Fred and Julie, each within the same AppDomain, both use the connection string **"Data Source=MySqlServer;Integrated Security=true"**, a connection pool group is created for the connection string, and two additional pools are created, one for Fred and one for Julie. If John and Martha use a connection string with an identical SQL Server login, **"Data Source=MySqlServer;User Id=lowPrivUser;Password=Strong?Password"** , then only a single pool is created for the **lowPrivUser** identity.
	Active connections: Indicates the number of connections that are currently in use.	Number	
	Free connections: Indicates the count of unused connections.	Number	Ideally, the value of this measure. A very low value indicates excessive connection usage.

	Inactive connection pools: Indicates the number of connection pools that have had no recent activity and are waiting to be disposed.	Number	
	Inactive connection pool groups: Indicates the number of inactive connection pool groups that were waiting to be deactivated i.e., to be pruned.	Number	
	Non-pooled connections: Indicates the number of active connections that are not using any of the connection pools.	Number	
	Pooled connections: Indicates the number of connections that are managed by the connection pooler.	Number	
	Reclaimed connections: Indicates the number of connections that have been reclaimed through garbage collection where Close or Dispose was not called by the application.	Number	Not explicitly closing or disposing connections hurts performance.
	Waiting connections: Indicates the number of connections that are currently awaiting completion of an action and are therefore unavailable for use by any other application.	Number	
	Soft connects: Indicates the rate at which connections are pulled from the connection pool.	Connects/Sec	
	Soft disconnects: Indicates the rate at which connections are returned to the connection pool.	Disconnects/Sec	

3.2.2 ASP .Net Oracle Data Provider Test

A data provider in the .NET Framework serves as a bridge between an application and a data source. A .NET Framework data provider enables you to return query results from a data source, execute commands at a data source, and propagate changes in a DataSet to a data source.

The Oracle Data Provider for .NET (ODP.NET) features optimized data access to the Oracle database from a .NET environment. ODP.NET allows developers to take advantage of advanced Oracle database functionality, including Real Application Clusters, XML DB, and advanced security. The data provider can be used from any .NET language, including C# and Visual Basic .NET.

This test reports many useful metrics that shed light on the health of the interactions between the ASP .Net sever and the Oracle database server.

Purpose	Sheds light on the health of the interactions between the ASP .Net sever and the Oracle database server		
Target of the test	The ASP .Net server		
Agent deploying the test	An internal agent		
Configurable parameters for the test	1. **TEST PERIOD** - How often should the test be executed 2. **HOST** - The host for which the test is to be configured 3. **PORT** - The port at which the specified host listens		
Outputs of the test	One set of results for the ASP .Net server being monitored		
Measurements made by the test	**Measurement**	**Measurement Unit**	**Interpretation**
	Hard connects: Indicates the number of actual connections per second that are being made to a database server.	Connects/Sec	
	Hard disconnects: Indicates the number of actual disconnects per second that are being made to a database server.	Disconnects/Sec	
	Active connection pool groups: Indicates the number of currently active connection pool groups.	Number	The value of this measure is controlled by the number of unique connection strings that are found in the AppDomain.

	Active connection pools: Indicates the number of currently active connection pools.	Number	When a connection is first opened, a connection pool is created based on matching criteria that associates the pool with the connection string in the connection. Each connection pool is associated with a distinct connection string. If the connection string is not an exact match to an existing pool when a new connection is opened, a new pool is created. Connections are pooled per process, per application domain, per connection string, and, when integrated security is used, per Windows identity. When using Windows Authentication (integrated security), both the *Active connection pool groups* and *Active connection pools* measures are significant. The reason is that connection pool groups map to unique connection strings. When integrated security is used, connection pools map to connection strings and additionally create separate pools for individual Windows identities. For example, if Fred and Julie, each within the same AppDomain, both use the connection string **"Data Source=MySqlServer;Integrated Security=true"**, a connection pool group is created for the connection string, and two additional pools are created, one for Fred and one for Julie. If John and Martha use a connection string with an identical SQL Server login, **"Data Source=MySqlServer;User Id=lowPrivUser;Password=Strong?Password"** , then only a single pool is created for the **lowPrivUser** identity.
	Active connections: Indicates the number of connections that are currently in use.	Number	
	Free connections: Indicates the count of unused connections.	Number	Ideally, the value of this measure. A very low value indicates excessive connection usage.

	Inactive connection pools: Indicates the number of connection pools that have had no recent activity and are waiting to be disposed.	Number	
	Inactive connection pool groups: Indicates the number of inactive connection pool groups that were waiting to be deactivated i.e., to be pruned.	Number	
	Non-pooled connections: Indicates the number of active connections that are not using any of the connection pools.	Number	
	Pooled connections: Indicates the number of connections that are managed by the connection pooler.	Number	
	Reclaimed connections: Indicates the number of connections that have been reclaimed through garbage collection where Close or Dispose was not called by the application.	Number	Not explicitly closing or disposing connections hurts performance.
	Waiting connections: Indicates the number of connections that are currently awaiting completion of an action and are therefore unavailable for use by any other application.	Number	
	Soft connects: Indicates the rate at which connections are pulled from the connection pool.	Connects/Sec	
	Soft disconnects: Indicates the rate at which connections are returned to the connection pool.	Disconnects/Sec	

3.2.3 ASP Sql Clients Test

This test reports metrics pertaining to client connections to the ASP .Net server.

Purpose	Reports metrics pertaining to client connections to the ASP .Net server		
Target of the test	An ASP .Net server		
Agent deploying the test	An internal agent		
Configurable parameters for the test	1. **TEST PERIOD** - How often should the test be executed 2. **HOST** - The host for which the test is to be configured 3. **PORT** - The port at which the specified host listens		
Outputs of the test	One set of results for the ASP .Net server being monitored		
Measurements made by the test	**Measurement**	**Measurement Unit**	**Interpretation**
	Connection pool size: Indicates the number of connection pools that have been created.	Number	If the connection pool maxes out while new connection requests are still coming in, you willl see connection requests refused, apparently at random. The cure in this case is simply to specify a higher value for the Max Pool Size property.
	Number of connections: Indicates the number of connections currently in the pool.	Number	
	Pooled connections: Indicates the number of connections that have been pooled.	Number	
	Pooled connections peak: Indicates the highest number of connections that have been used.	Number	If the value of this measure is at the Max Pool Size value, and the value of the *Failed connects* measure increases while the application is running, you might have to consider increasing the size of the connection pool.
	Failed connects: Indicates the number of connection attempts that have failed.	Number	If the connection pool maxes out while new connection requests are still coming in, you willl see connection requests refused, apparently at random. The cure in this case is simply to specify a higher value for the Max Pool Size property.

3.2.4 ASP .Net Workers Test

The AspNetWorkerTest reports statistics pertaining to the performance of the worker process of the ASP .Net server.

Purpose	Reports statistics pertaining to the performance of the worker process of the ASP .Net servers		
Target of the test	The ASP .Net server		
Agent deploying the test	An internal agent		
Configurable parameters for the test	1. **TEST PERIOD** - How often should the test be executed 2. **HOST** - The host for which the test is to be configured 3. **PORT** - The port at which the specified host listens		
Outputs of the test	One set of results for the ASP .Net server being monitored		
Measurements made by the test	**Measurement**	**Measurement Unit**	**Interpretation**
	Application restarts: The number of application restarts.	Number	In a perfect world, the application domain will and should survive for the life of the process. Even if a single restart occurs, it is a cause for concern because proactive and reactive restarts cause automatic recycling of the worker process. Moreover, restarts warrant recreation of the application domain and recompilation of the pages, both of which consume a lot of time. To investigate the reasons for a restart, check the values set in the processModel configuration.
	Applications running: The number of applications currently running.	Number	
	Requests current: The number of requests currently handled by the ASP.NET ISAPI. This includes those that are queued , executing, or waiting to be written to the client.	Number	

	Request execution time: The number of seconds taken to execute the last request.	Number	In version 1.0 of the framework, the execution time begins when the worker process receives the request, and stop when the ASP.NET ISAPI sends HSE_REQ_DONE_WITH_SESSION to IIS. In version 1.1 of the framework, execution begins when the HttpContext for the request is created, and stop before the response is sent to IIS. The value of this measure should be stable. Any sudden change from the previous recorded values should be notified.
	Requests queued: The number of requests currently queued.	Number	When running on IIS 5.0, there is a queue between inetinfo and aspnet_wp, and there is one queue for each virtual directory. When running on IIS 6.0, there is a queue where requests are posted to the managed ThreadPool from native code, and a queue for each virtual directory. This counter includes requests in all queues. The queue between inetinfo and aspnet_wp is a named pipe through which the request is sent from one process to the other. The number of requests in this queue increases if there is a shortage of available I/O threads in the aspnet_wp process. On IIS 6.0 it increases when there are incoming requests and a shortage of worker threads.
	Requests rejected: The number of rejected requests	Number	Requests are rejected when one of the queue limits is exceeded. An excessive value of this measure hence indicates that the worker process is unable to process the requests due to overwhelming load or low memory in the processor.
	Requests wait time: The number of seconds that the most recent request spent waiting in the queue, or named pipe that exists between inetinfo and aspnet_wp. This does not include any time spent waiting in the application queues.	Secs	

	Worker processes running: The current number of aspnet_wp worker processes	Number	Every application executing on the .NET server corresponds to a worker process. Sometimes, during active or proactive recycling, a new worker process and the worker process that is being replaced may coexist. Under such circumstances, a single application might have multiple worker processes executing for it. Therefore, if the value of this measure is not the same as that of *Applications running*, then it calls for closer examination of the reasons behind the occurence.
	Worker process restarts: The number of aspnet_wp process restarts in the machine	Number	Process restarts are expensive and undesirable. The values of this metric are dependent upon the process model configuration settings, as well as unforeseen access violations, memory leaks, and deadlocks.

3.2.5 ASP .Net Applications Test

NET introduces the concept of an application domain, or AppDomain. Like a process, the AppDomain is both a container and a boundary. The .NET runtime uses an AppDomain as a container for code and data, just like the operating system uses a process as a container for code and data. As the operating system uses a process to isolate misbehaving code, the .NET runtime uses an AppDomain to isolate code inside of a secure boundary. An AppDomain belongs to only a single process, but a single process can hold multiple AppDomains.

An AppDomain is relatively cheap to create (compared to a process), and has relatively less overhead to maintain than a process. For these reasons, an AppDomain is a great solution for the ISP who is hosting hundreds of applications. Each application can exist inside an isolated AppDomain, and many of these AppDomains can exist inside of a single process.

From this, it can be inferred that to understand how well an application is performing, it is necessary to monitor the AppDomain in which that application exists. This involves the following:

- Tracking the requests to the AppDomain, measuring the time taken by the AppDomain to service the requests, and keeping an eye on the pending requests count on the AppDomain;
- Understanding how the AppDomain uses its cache and isolating irregularities in cache usage or sizing;
- Monitoring the session load on the AppDomain and measuring the efficiency of the AppDomain in handling this load;
- Capturing errors encountered by the AppDomain.

The **ASP .Net Applications** test performs all of the above for every AppDomain on the monitored ASP .Net server. This way, the test proactively alerts administrators to potential request processing bottlenecks in any AppDomain, promptly identifies the AppDomain that is utilizing its cache ineffectively, instantly captures errors (if any) in the AppDomain and brings them to the notice of administrators, and rapidly notifies administrators if any AppDomain is overloaded with sessions or is abandoning/timing out sessions at a brisk pace.

Purpose	Proactively alerts administrators to potential request processing bottlenecks in any AppDomain, promptly identifies the AppDomain that is utilizing its cache ineffectively, instantly captures errors (if any) in the AppDomain and brings them to the notice of administrators, and rapidly notifies administrators if any AppDomain is overloaded with sessions or is abandoning/timing out sessions at a brisk pace		
Target of the test	An IIS web server		
Agent deploying the test	An internal agent		
Configurable parameters for the test	1. **TEST PERIOD** - How often should the test be executed 2. **HOST** - The host for which the test is to be configured 3. **PORT** - The port at which the specified host listens		
Outputs of the test	One set of results for every ASP .Net application/application domain on a monitored ASP .Net server		
Measurements made by the test	**Measurement**	**Measurement Unit**	**Interpretation**
	Cache total entries: The current number of entries in the cache (both User and Internal) of this AppDomain/application.	Number	
	Cache hit ratio: The current hit-to-miss ratio of all cache requests (both user and internal) to the cache of this AppDomain/application.	Percent	Physical I/O takes a significant amount of time, and also increases the CPU resources required. The server configuration should therefore ensure that the required information is available on the memory. A low value of this measure indicates that physical I/O is greater.
	Cache turnover rate The number of additions and removals per second (both user and internal) to the cache of this AppDomain/application.	Cached/Sec	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 of this AppDomain/application.	Number	
	Cache user hit ratio: Total hit-to-miss ratio of requests to the user cache of this AppDomain/application .	Percent	A high value of this measure is indicative of the good health of the server.

	Cache user turnover rate: The number of additions and removals per second to the user cache of this AppDomain/application.	Cached/Sec	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 of this AppDomain/application.	Number	
	Output cache hit ratio: The total hit-to-miss ratio of requests to the Output Cache of this AppDomain/application.	Percent	A high value of this measure is a sign of good health.
	Output cache turnover rate: The number of additions and removals per second to the Output Cache of this AppDomain/application.	Cached/Sec	Output caching allows you to store dynamic page and user control responses on any HTTP 1.1 cache-capable device in the output stream, from the originating server to the requesting browser. On subsequent requests, the page or user control code is not executed; the cached output is used to satisfy the request Sudden increases in the value of this measure are indicative of backend latency.
	Compilation total: The total number of compilations that have taken place during the lifetime of the current Web server process. This occurs when a file with a .aspx, .asmx, asax,.ascx, or .ashx extension or code-behind source files are dynamically compiled on the server.	Number	
	Processing errors: The rate at which configuration and parsing errors occur on this AppDomain/application.	Errors/Sec	A consistent increase in the value of this measure could prove to be fatal for the application domain.

	Compilation errors: The rate at which compilation errors occur on this AppDomain/application.	Errors/Sec	The response is cached, and this counter increments only once until recompilation is forced by a file change.
	Runtime errors: The rate at which run-time errors occur on this AppDomain/application.	Errors/Sec	
	Unhandled runtime errors: The rate of unhandled runtime exceptions on this AppDomain/application.	Errors/Sec	A consistent increase in the value of this measure could prove to be fatal for the application domain. This measure however, does not include the following: • Errors cleared by an event handler (for example, by Page_Error or Application_Error) • Errors handled by a redirect page • Errors that occur within a try/catch block
	Requests executing: The number of requests currently executing on this AppDomain/application.	Number	This measure is incremented when the HttpRuntime begins to process the request and is decremented after the HttpRuntime finishes the request.
	Requests app queue: The number of requests currently in the application request queue of this AppDomain/application.	Number	A steady increase in the value of this measure could indicate a request processing bottleneck in the AppDomain.
	Requests not found: The number of requests that did not find the required resource on this AppDomain/application.	Number	Ideally, the value of this measure should be 0.
	Requests not authorized: The number of requests to this AppDomain/application that failed due to unauthorized access.	Number	Values greater than 0 indicate that proper authorization has not been provided, or invalid authors are trying to access a particular resource.

	Requests timed out: The number of requests to this AppDomain/application that timed out.	Number	Ideally, the value of this measure should be 0.
	Requests succeeded: The rate at which requests to this AppDomain/application succeeded.	Requests/Sec	
	Request rate: Indicates the number of requests executed per second by this AppDomain/application.	Number	This represents the current throughput of the application.
	Pipeline instances: Indicates the number of active pipeline instances for this AppDomain/application.	Number	Since only one execution thread can run within a pipeline instance, this number gives the maximum number of concurrent requests that are being processed for a given application. Ideally, the value of this measure should be low.
	Number of errors: Indicates the total sum of all errors that occurred during the execution of HTTP requests by this AppDomain/application.	Number	This measure should be kept at 0 or a very low value.
	SQL connections: Indicates the number of connections to the SQL Server used by session state.	Number	An unusually high value may indicate a sudden increase in sessions to the SQL Server.
	State server connections: Indicates the number of connections to the StateServer used by session state.	Number	An unusually high value may indicate a sudden increase in sessions to the StateServer.
	Abandoned ASPNet application sessions: Indicates the number of sessions on this AppDomain/application that have been explicitly abandoned during the last measurement period.	Number	Ideally, the value of this measure should be 0.

	Active ASPNet application sessions: Indicates the currently active sessions on this AppDomain/application.	Number	This is a good indicator of the current session load on the AppDomain/application.
	Timedout ASPNet application sessions: Indicates the number of sessions on this AppDomain/application that timed out during the last measurement period.	Number	Ideally, the value of this measure should be 0.
	ASPNet application sessions: Indicates the total number of sessions on this AppDomain/application during the last measurement period.	Number	
	Request execution time: The number of seconds this AppDomain/application took to execute the last request.	Number	In version 1.0 of the framework, the execution time begins when the worker process receives the request, and stop when the ASP.NET ISAPI sends HSE_REQ_DONE_WITH_SESSION to IIS. In version 1.1 of the framework, execution begins when the HttpContext for the request is created, and stop before the response is sent to IIS. The value of this measure should be stable. Any sudden change from the previous recorded values should be notified.

3.2.6 ASP .NET CLR Test

The **Common Language Runtime** (**CLR**), the virtual machine component of Microsoft's .NET framework, manages the execution of .NET programs. A process known as just-in-time (JIT) compilation converts compiled code into machine instructions which the computer's CPU then executes. The CLR provides additional services including memory management, type safety, exception handling, garbage collection, security and thread management. All programs written for the .NET framework, regardless of programming language, are executed by the CLR.

This is why, if a application running on the .NET framework slows down, the first place that an administrator should look for hints is the CLR. Some of the common causes of poor program/application are excessive CLR heap usage, ineffective garbage collection by CLR, class loading failures, thread contentions on the CLR, and expensive run time checks by CLR. To be able to capture such anomalies on-the-fly, it is imperative that administrators closely monitor the performance of the CLR. The **ASP .NET CLR** test helps administrators do just that! The test auto-discovers the worker processes running in the ASP .NET server, and for every process, reports the following:

- The count of CLR exceptions that were thrown by that worker process;

- The heap memory usage of the worker process; in the event of excessive usage, you can also take a look at the garbage collection-related statistics reported by this test for that worker process to determine whether/not ineffective GC is causing heap memory usage to soar.
- The count of classes loaded and class loading failures;
- The rate of thread contentions in the worker process;
- The count of waiting threads;
- The time spent in performing runtime Code Access Security (CAS) checks;
- The count of runtime checks performed.

These metrics provide useful pointers to the source of performance degradations experienced by applications.

<table>
<tr><td>Purpose</td><td colspan="3">Auto-discovers the worker processes running in the ASP .NET server, and for every process, reports the following:
The count of CLR exceptions that were thrown by that worker process;
The heap memory usage of the worker process; in the event of excessive usage, you can also take a look at the garbage collection-related statistics reported by this test for that worker process to determine whether/not ineffective GC is causing heap memory usage to soar.
The count of classes loaded and class loading failures;
The rate of thread contentions in the worker process;
The count of waiting threads;
The time spent in performing runtime Code Access Security (CAS) checks;
The count of runtime checks performed.
These metrics provide useful pointers to the source of performance degradations experienced by applications.</td></tr>
<tr><td>Target of the test</td><td colspan="3">An IIS web server</td></tr>
<tr><td>Agent deploying the test</td><td colspan="3">An internal agent</td></tr>
<tr><td>Configurable parameters for the test</td><td colspan="3">1. TEST PERIOD - How often should the test be executed
2. HOST - The host for which the test is to be configured
3. PORT - The port at which the specified host listens</td></tr>
<tr><td>Outputs of the test</td><td colspan="3">One set of results for every worker process on the monitored ASP .Net server</td></tr>
<tr><td rowspan="2">Measurements made by the test</td><td>Measurement</td><td>Measurement Unit</td><td>Interpretation</td></tr>
<tr><td>Clr exceptions:
The total number of managed exceptions thrown per second.</td><td>Exceptions/Sec</td><td>Exceptions are very costly and can severely degrade your application performance. A high value of this measure is therefore an indicator of potential performance issues.</td></tr>
</table>

	Heap mem usage: The number of bytes committed by managed objects. This is the sum of the large object heap and the generation 0, 1, and 2 heaps.	MB	
	Gen 0 collections: The rate at which the generation 0 objects (youngest; most recently allocated) are garbage collected (Gen 0 GC) since the start of the application.	Collections/Sec	
	Gen 1 collections: The rate at which the generation 1 objects have been garbage collected since the start of the application. Objects that survive are promoted to generation 2.	Collections/Sec	
	Gen 2 collections: The number of seconds taken to execute the last request.	Number	The number of times generation 2 objects have been garbage collected since the start of the application. Generation 2 is the highest, thus objects that survive collection remain in generation 2. Gen 2 collections can be very expensive, especially if the size of the Gen 2 heap is huge.
	Time in gc: % Time in GC is the percentage of elapsed time that was spent in performing a garbage collection (GC) since the last GC cycle.	Percent	This measure is usually an indicator of the work done by the Garbage Collector on behalf of the application to collect and conserve memory. This measure is updated only at the end of every GC and the measure reflects the last observed value; its not an average.
	Classes loaded: Indicates the cumulative number of classes loaded in all assemblies since the start of this worker process.	Number	A class is essentially the blueprint for an object. It contains the definition for how a particular object will be instantiated at runtime, such as the properties and methods that will be exposed publicly by the object and any internal storage structures.
	Current classes loaded: Indicates the current number of classes loaded in all Assemblies.	Number	An unusually high value may indicate a sudden increase in classes which loaded on to this .NET application.

	Rate of assemblies: The rate at which Assemblies were loaded.	Assembles/Sec	Also known as Managed DLLs, assemblies are the fundamental unit of deployment for the .NET platform. The .NET Framework itself is made up of a number of assemblies, including mscorlib.dll, among others. The assembly boundary is also where versioning and security are applied. An assembly contains Intermediate Language generated by a specific language compiler, an assembly manifest (containing information about the assembly), type metadata, and resources. If the Assembly is loaded as domain-neutral from multiple AppDomains then this counter is incremented once only. Assemblies can be loaded as domain-neutral when their code can be shared by all AppDomains or they can be loaded as domain-specific when their code is private to the AppDomain. This counter is not an average over time; it displays the difference between the values observed in the last two samples divided by the duration of the sample interval.
	Rate of classes loaded: This rate at which the classes loaded in all Assemblies.	Classes/Sec	This counter is not an average over time; it displays the difference between the values observed in the last two samples divided by the duration of the sample interval.
	Rate of load failures: The rate of load failures on this worker process.	Failures/Sec	This counter is not an average over time; it displays the difference between the values observed in the last two samples divided by the duration of the sample interval. These load failures could be due to many reasons like inadequate security or illegal format.
	Current appdomains: The number of AppDomains currently loaded in this worker process.	Number	AppDomains (application domains) provide a secure and versatile unit of processing that the CLR can use to provide isolation between applications running in the same process.
	Current assemblies: The number of assemblies currently loaded on this worker process.	Number	If the Assembly is loaded as domain-neutral from multiple AppDomains then this counter is incremented once only. Assemblies can be loaded as domain-neutral when their code can be shared by all AppDomains or they can be loaded as domain-specific when their code is private to the AppDomain.
	Loader heap size: The size of the memory committed by the class loader.	MB	Committed memory is the physical memory for which space has been reserved on the disk paging file.

	Load failures: The number of classes that have failed to load during the last measurement period.	Number	These load failures could be due to many reasons like inadequate security or illegal format.
	Appdomains loaded: The number of AppDomains loaded during the last measurement period.	Number	
	Num assemblies: The number of assemblies loaded during the last measurement period.	Number	An assembly in ASP.NET is a collection of single-file or multiple files. The assembly that has more than one file contains either a dynamic link library (DLL) or an EXE file. The assembly also contains metadata that is known as assembly manifest. The assembly manifest contains data about the versioning requirements of the assembly, author name of the assembly, the security requirements that the assembly requires to run, and the various files that form part of the assembly. The biggest advantage of using ASP.NET Assemblies is that developers can create applications without interfering with other applications on the system.
	Current logical threads: The number of current managed thread objects in this worker process. This measure maintains the count of both running and stopped threads.	Number	
	Current physical threads: The number of native operating system threads created and owned by the common language runtime to act as underlying threads for managed thread objects. This measure does not include the threads used by the runtime in its internal operations.	Number	

	Current recognized threads: The number of threads that are currently recognized by the runtime. These threads are associated with a corresponding managed thread object.	Number	
	Contention rate: The rate at which threads in the runtime attempt to acquire a managed lock unsuccessfully.	Rate/Sec	
	Current queue length: The total number of threads that are currently waiting to acquire a managed lock in the application.	Number	
	Queue length rate: Indicates the rate at which threads are waiting to acquire some lock.	Threads/Sec	
	Recognized threads rate: Indicates the number of threads per second that have been recognized by the CLR.	Threads/Sec	The recognized threads have a corresponding .NET thread object associated with them. These threads are not created by the CLR; they are created outside the CLR but have since run inside the CLR at least once. Only unique threads are tracked; threads with the same thread ID re-entering the CLR or recreated after thread exit are not counted twice.
	Queue length peak: Indicates the total number of threads that waited to acquire some managed lock during the last measurement period.	Number	A high turnover rate indicates that items are being quickly added and removed, which can be expensive.
	Recognized threads: Indicates the total number of threads that have been recognized by the CLR during the last measurement period.	Number	The recognized threads have a corresponding .NET thread object associated with them. These threads are not created by the CLR; they are created outside the CLR but have since run inside the CLR at least once. Only unique threads are tracked; threads with the same thread ID re-entering the CLR or recreated after thread exit are not counted twice.

	Contention threads: Indicates the total number of times threads in the CLR have attempted to acquire a managed lock unsuccessfully.	Number	Managed locks can be acquired in many ways; by the lock statement in C# or by calling System.Monitor.Enter or by using MethodImplOptions.Synchronized custom attribute.
	Time in runtime checks: Indicates the percentage of elapsed time spent in performing runtime Code Access Security (CAS) checks during the last measurement period.	Percent	If this counter is high, revisit what is being checked and how often. The application may be executing unnecessary stack walk depths. Another cause for a high percentage of time spent in runtime checks could be numerous linktime checks.
	Stack walk depth: Indicates the depth of the stack during that last measurement period.	Number	
	Link time checks: Indicates the total number of linktime Code Access Security (CAS) checks during the last measurement period.	Number	The value displayed is not indicative of serious performance issues, but it is indicative of the health of the security system activity.
	Runtime checks: Indicates the total number of runtime CAS checks performed during the last measurement period.	Number	A high number for the total runtime checks along with a high stack walk depth indicates performance overhead.

3.3 The Web Server Layer

Besides the **WebServer**, **Http**, and **HttpPost** tests that have already been dealt with in Chapter 2 of this document, the **Web Server** layer of an IIS web server (see Figure 3.8), is additionally mapped to an **IIS Web Transactions** test, an **IISWebCache** test, an **IISWebSites** test, an **IISApplicationPools** test, a **Windows Process Activation Service** test, and a **Worker Processes Statistics** test.

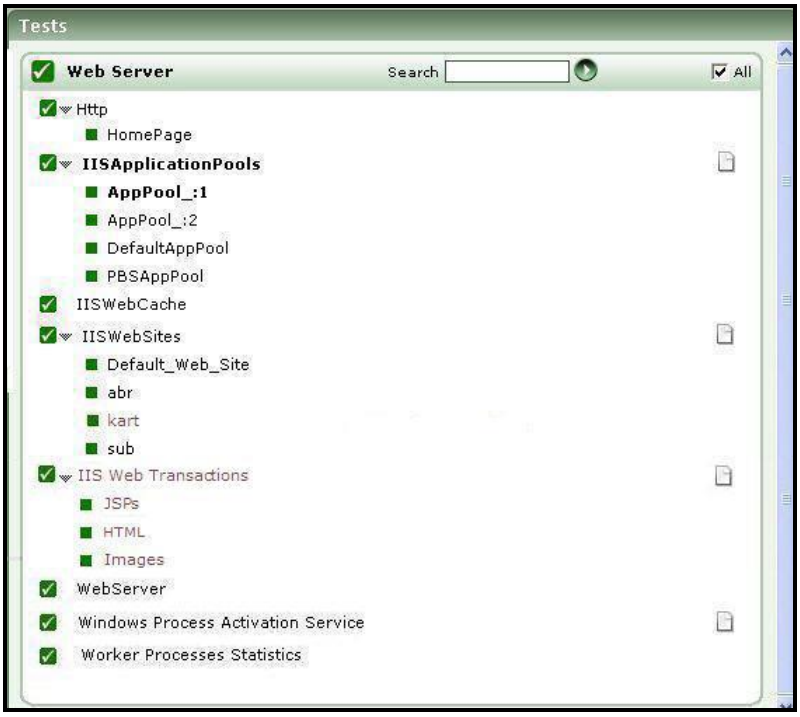


Figure 3.8: Tests associated with the Web Server layer of the IIS web server

3.3.1 Web Server Test

This internal test complements the measurements made from an external perspective by the Http test. It collects various statistics that measure the request load on the web server and its responsiveness to the load. The statistics collected by this test are detailed below:

Purpose	To measure the state of a web server		
Target of the test	A web server instance		
Agent deploying the test	An internal agent		
Configurable parameters for the test	1. **TEST PERIOD** - How often should the test be executed 2. **HOST -** The host for which the test is to be configured 3. **PORT -** the port to which the specified **HOST** listens		
Outputs of the test	One set of results for every web server monitored		
Measurements made by the test	**Measurement**	**Measurement Unit**	**Interpretation**
	Connections: Rate of connections to the web server.	Conns/Sec	An increase or decrease in connection rate can represent a change in user workload

	Requests: Rate of requests to the web server during the last measurement period.	Reqs/Sec	With the advent of HTTP/1.1, multiple requests can be transmitted over the same TCP connection. The ratio of requests per connection can provide an idea of the effectiveness of the HTTP 1.1 protocol.
	Data transmitted: Rate at which the data was transmitted by the server during the last measurement period.	KB/Sec	A large increase in the data transmission rate can be indicative of an increase in the popularity of one or more web sites hosted on the server.
	Data received: Rate at which the data was received by the server during the last measurement period.	KB/Sec	An increase in this value is indicative of an increase in user requests to the server.
	Errors: Percentage of error responses from the server during the last measurement period.	Percent	Percentage of responses with a 400 or 500 status code.
	400 errors: Percentage of responses with a status code in the range 400-499 during the last measurement period.	Percent	A high value indicates a number of missing/error pages.
	Current requests: Number of server threads/processes currently in use for serving requests (this measurement is not available for Apache web servers).	Number	If a majority of the server threads/processes are in use simultaneously, this may be indicative of a server bottleneck.

Note:

The WebServer test will report metrics for an IIS web server executing on a Windows 2008 host, only if the **IIS Management Scripts and Tools** feature is installed on that host, and the **Web Server** role you create enables this feature. If the aforesaid feature is not enabled for the **Web Server** role, then remove the role and re-create it with the feature enabled.

3.3.2 IIS Web Sites Test

Typically, the WebSiteTransaction test monitors only those web sites (on an IIS web server) for which transactions have been explicitly configured using the eG administrative interface. Therefore, if an IIS web server hosts multiple sites, and if transactions have been configured only for one of them, then the WebSiteTransaction test will report statistics pertaining to that one site alone. Performance issues in other web sites, will hence go undetected and consequently, unresolved. Using the IISWebSites test, you can monitor each of the web sites hosted by an IIS web server, regardless of whether or not transactions have been configured for them.

Purpose	Reports statistics pertaining to every web site running on an IIS web server		
Target of the test	A web site supported by a web server		
Agent deploying the test	An internal agent		
Configurable parameters for the test	1. **TEST PERIOD** - How often should the test be executed 2. **HOST -** The host for which the test is to be configured 3. **PORT -** The port to which the specified **HOST** listens 4. **WEBSITES** - By default, 'all' is displayed here, indicating that by default, this test reports statistics related to all the web sites on the web server. To monitor a particular web site, provide that web site's name in this text box. Multiple web sites can be specified as a comma-separated list. While providing web site names, ensure that the specification exactly matches (even in case) the instance names of the performance object, 'Web Service'.		
Outputs of the test	One set of results for every web site monitored		
Measurements made by the test	**Measurement**	**Measurement Unit**	**Interpretation**
	Availability: Indicates whether the web site is currently available or not.	Percent	The value 0 indicates that the website is not running, the value 100 indicates that it is up and running.
	Data receive rate: Indicates the rate at which the data is received by the web server.	KBytes/Sec	An increase in this value is indicative of an increase in user requests to the web server.

	Data transmit rate: Indicates the rate at which the data is transmitted by the web server.	KBytes/Sec	A large increase in the data transmission rate can be indicative of an increase in the popularity of one or more web sites hosted on the server.
	Connection rate: Indicates the rate of connections to the web server.	Conns/Sec	An increase or decrease in connection rate can represent a change in user workload.
	Current requests: Indicates the number of server threads/ processes that are currently in use for serving requests.	Number	If a majority of the server threads/processes are in use simultaneously, this may be indicative of a server bottleneck.
	Requests serviced rate: Indicates the rate at which all HTTP Requests are issued.	Requests/Sec	
	Current anonymous users: Indicates the number of users who currently have an anonymous connection.	Number	
	Current non-anonymous users: Indicates the number of users who currently have non-anonymous connection.	Number	
	Get requests: Indicates the rate at which HTTP requests using the GET method are made.	Requests/Sec	
	Post requests: Indicates the rate at which HTTP requests using the POST method are made.	Requests/Sec	
	Head requests: Indicates the rate at which HTTP requests using the HEAD method are issued.	Requests/Sec	
	Maximum connections: Indicates the maximum number of simultaneous connections established.	Number	

	Not found requests: Indicates the rate of errors due to requests that could not be satisfied by the web server because the requested document could not found.	Requests/Sec	A high value indicates a number of missing/error pages.
	Requests serviced: Indicates the number of HTTP requests serviced, currently.	Number	

3.3.3 IIS Web Cache Test

This test monitors the IIS web server's web cache and reports critical performance metrics pertaining to it.

Note that this test will not work on Windows 2000.

Purpose	Monitors the IIS web server's web cache and reports critical performance metrics pertaining to it		
Target of the test	A web site supported by a web server		
Agent deploying the test	An internal agent		
Configurable parameters for the test	1. **TEST PERIOD** - How often should the test be executed 2. **HOST -** The host for which the test is to be configured 3. **PORT -** The port to which the specified **HOST** listens		
Outputs of the test	One set of results for every web site monitored		
Measurements made by the test	**Measurement**	**Measurement Unit**	**Interpretation**
	File cache memory usage: Indicates data used for the user-mode file cache.	KB	An unusually high value may indicate a sudden increase of memory usage in file cache.
	Files cached: Indicates the number of files, the content of which is in the user-mode cache.	Number	
	Metadata cached: Indicates the metadata information blocks currently in the user-mode cache.	Number	

	URIs cached: Indicates the number of URI information blocks in the user-mode cache.	Number	
	File cache hits: Indicates the number of successful lookups in the user-mode file cache during the last measurement period.	Number	A consistent decrease in the value of this measure is indicative of an increase in direct disk accesses. This is a cause for concern.
	File cache misses: Indicates the number of unsuccessful lookups in the user-mode file cache during the last measurement period.	Number	A consistent increase in the value of this measure is indicative of an increase in direct disk accesses. This is a cause for concern.
	File cache hit ratio: Indicates the ratio between the file cache hits and file cache misses.	Percent	A very low percentage indicates that files requested are being directly accessed from the disk and not from the file cache.

3.3.4 Application Pool Workers Test

An *application pool* is a group of one or more URLs that are served by a worker process or a set of worker processes. Application pools set boundaries for the applications/web sites they contain, which means that any applications/web sites that are running outside a given application pool cannot affect the applications/web sites in the application pool. On the other hand, a single resource-intensive application/web site within a pool can affect overall pool performance, and eventually impact web server performance adversely! This poses a huge management challenge, particularly in MSP environments, where a single web server may host multiple web sites – one each for every MSP customer. A web server slowdown in such environments often leaves administrators flummoxed, as they struggle to fathom what problem in which web site of which application pool is responsible for the slowdown. This is where the **Application Pool Workers** test helps!

By periodically monitoring the overall status, resource usage, open handles, and I/O activity in each application pool on an IIS web server, the test quickly and accurately points you to the problematic application pool and the nature of its problem – is it because the application pool has stopped? is it excessive resource usage by a web site/application in the pool? is it a handle leak? is it an I/O processing bottleneck? You can even quickly drill down to the exact web site/application that is causing this problem, using the detailed diagnosis capability of the test. In addition, the test also captures and reports operational issues with the worker processes that service requests to an application pool, thereby shedding light on the contribution of worker processes to slowdowns/outages that a web server suffers.

Purpose	Periodically the overall status, resource usage, open handles, and I/O activity in each application pool on an IIS web server; in addition, the test also captures and reports operational issues with the worker processes that service requests to an application pool, thereby shedding light on the contribution of worker processes to slowdowns/outages that a web server suffers.
Target of the	IIS web server

<table>
<tr><td>test</td><td colspan="3"></td></tr>
<tr><td>Agent deploying the test</td><td colspan="3">An internal agent</td></tr>
<tr><td>Configurable parameters for the test</td><td colspan="3">1. TEST PERIOD - How often should the test be executed
2. HOST - The host for which the test is to be configured
3. PORT - The port to which the specified HOST listens
4. DETAILED DIAGNOSIS - To make diagnosis more efficient and accurate, the eG Enterprise suite embeds an optional detailed diagnostic capability. With this capability, the eG agents can be configured to run detailed, more elaborate tests as and when specific problems are detected. To enable the detailed diagnosis capability of this test for a particular server, choose the On option. To disable the capability, click on the Off option. The option to selectively enabled/disable the detailed diagnosis capability will be available only if the following conditions are fulfilled:
• The eG manager license should allow the detailed diagnosis capability
• Both the normal and abnormal frequencies configured for the detailed diagnosis measures should not be 0.</td></tr>
<tr><td>Outputs of the test</td><td colspan="3">One set of results for every application pool on the IIS web server monitored</td></tr>
<tr><td>Measurements made by the</td><td>Measurement</td><td>Measurement Unit</td><td>Interpretation</td></tr>
</table>

<table>
<tr>
<td rowspan="2">test</td>
<td>Application pool status:

Indicates the current status of this application pool.</td>
<td></td>
<td>The values that this measure can report and their corresponding numeric values have been listed in the table below:

<table>
<tr><th>Measure Value</th><th>Numeric Value</th></tr>
<tr><td>Stopped</td><td>0</td></tr>
<tr><td>Started</td><td>1</td></tr>
</table>

If this measure reports the value Sopped, then you need to make sure that the application pool is started. If a pool has stopped, then you can use the detailed diagnosis of this measure to identify the applications/web sites that are impacted as a result of this pool stopping.

This measure is reported for IIS v6 (and above) only.

Note:

By default, the measure reports the Measure Values listed in the table above to indicate the status of the application pool. The graph of this measure however represents the same using the numeric equivalents only.</td>
</tr>
<tr>
<td>Pool restarted

Indicates whether this pool is restarted or not.</td>
<td></td>
<td>The values that this measure can report and their corresponding numeric values have been listed in the table below:

<table>
<tr><th>Measure Value</th><th>Numeric Value</th></tr>
<tr><td>No</td><td>0</td></tr>
<tr><td>Yes</td><td>1</td></tr>
</table>

This measure is reported for IIS v7 (and above) only.

Note:

By default, the measure reports the Measure Values listed in the table above to indicate the restart status of the application pool. The graph of this measure however represents the same using the numeric equivalents only.</td>
</tr>
</table>

	Number of processes: Indicates the number of worker processes that are currently servicing requests to this application pool.	Number	A zero value in this measure indicates that the application pool has crashed. Use the detailed diagnosis of this measure to know the **PID** of every worker process that is handling requests to this pool and the command line argument revealing the name of the application/web site in the application pool, which is accessed by that worker process. In addition, the detailed diagnosis also displays the resource allocations (private data, pool paged data, non pool paged data), the resource usage (CPU and memory), open handle count, thread count, and I/O activity of each application in the pool that is accessed by every worker process. This way, resource-intensive applications in the pool can be isolated. **This measure is reported for IIS v6 (and above) only.**
	CPU utilization: Indicates the percentage CPU utilization of this application pool.	Percent	A higher value indicates excessive CPU utilization. **This measure is reported for IIS v6 (and above) only.**
	Memory utilization: Indicates the percentage of total memory utilized by this application pool.	Percent	If the value of this measure consistently increases, it indicates a memory bottleneck. **This measure is reported for IIS v6 (and above) only.**
	Number of threads: Indicates the number of threads that are currently active in this application pool.	Number	This is an indicator of the workload on the pool. **This measure is reported for IIS v6 (and above) only.**
	Number of handles: Indicates the number of handles currently opened by this application pool.	Number	A sudden/consistent increase in the value of this measure could indicate a handle leak in the pool. **This measure is reported for IIS v6 (and above) only.**
	Page faults: Indicates the rate of page faults happening in this application pool.	Faults/Sec	**This measure is reported for IIS v6 (and above) only.**

	Private data: Indicates the amount of data that this application pool has currently allocated and cannot be shared with other application pools.	MB	A gradual growth in this measure indicates a memory leak in the websites that are running in this application pool. **This measure is reported for IIS v6 (and above) only.**
	Pool paged data: Indicates the amount of memory currently allocated from the paged pool to this application pool. The Paged Pool is an area of the System's virtual memory that is limited in size and used for various system related functions.	MB	A gradual growth in this measure indicates a memory leak in the websites that are running in the application pool. **This measure is reported for IIS v6 (and above) only.**
	Pool non paged data: Indicates the amount of memory currently allocated from the non-paged pool to this application pool. The Non-Paged Pool is an area of the System's virtual memory that is limited in size and used for kernel and device driver functions.	MB	A gradual growth in this measure indicates a memory leak in the websites that are running in the application pool. Running out of space in the nonpaged pool often results in a Blue Screen. **This measure is reported for IIS v6 (and above) only.**
	I/O reads: Indicates the rate at which the worker process(es) servicing requests to this application pool is reading data.	KBytes/Sec	**These measures are reported for IIS v6 (and above) only.** A very low or a consistent dip in the value of these measures is a cause for concern, as it could indicate a bottleneck when processing read requests to the application pool. You can compare the value of these measures across pools, to identify that application pool that is the slowest in read request processing.
	I/O read operations: Indicates the rate at which the worker process(es) serving requests to this pool is performing I/O reads.	Operations/Sec	
	I/O writes: Indicates the rate at which the worker process(es) serving requests to this pool is writing data.	KBytes/Sec	**These measures are reported for IIS v6 (and above) only.** A very low or a consistent dip in the value of these measures is a cause for concern, as it could indicate a bottleneck when processing write requests to the application pool. You can compare the value of these measures across pools, to identify that application pool that is the slowest in write request processing.
	I/O write operations: Indicates the rate at which the worker process(es) serving requests to this pool is issuing write I/O commands.	Operations/Sec	

	Pool uptime: Indicates the length of time, in seconds, that this application pool has been running since it was started.	Secs	
	Current process: Indicates the number of worker processes currently servicing requests to this application pool.	Number	**This measure is reported for IIS v7 (and above) only.**
	Recent failure: Indicates the number of times the worker processes associated with this pool failed during the rapid-fail protection interval.	Number	Rapid Fail Protection is a feature in IIS 7, which is enabled by default. This feature checks for a specified number of hard failures in a given time period - 5 failures within 5 minutes by default. If a hard failure that occurs meets this default setting, then the Application Pool will crash and does not automatically restart. It would be good practice to set the hard failure setting as the threshold for this measure; this way, you can be proactively alerted to a potential application pool crash. **This measure is reported for IIS v7 (and above) only.**
	Pool recycle Indicates the number of times this application pool was recycled since Windows Process Activation Service (WAS) started.	Number	If you have a problematic application and you cannot easily correct the code that causes the problems, you can limit the extent of these problems by periodically recycling the worker process that services the application. In addition to recycling an application pool on demand when problems occur, you can configure an application pool to recycle a worker process for the following reasons: k. At a scheduled time l. After an elapsed time m. After reaching a number of requests n. After reaching a virtual memory threshold o. After reaching a used memory threshold **This measure is reported for IIS v7 (and above) only.**

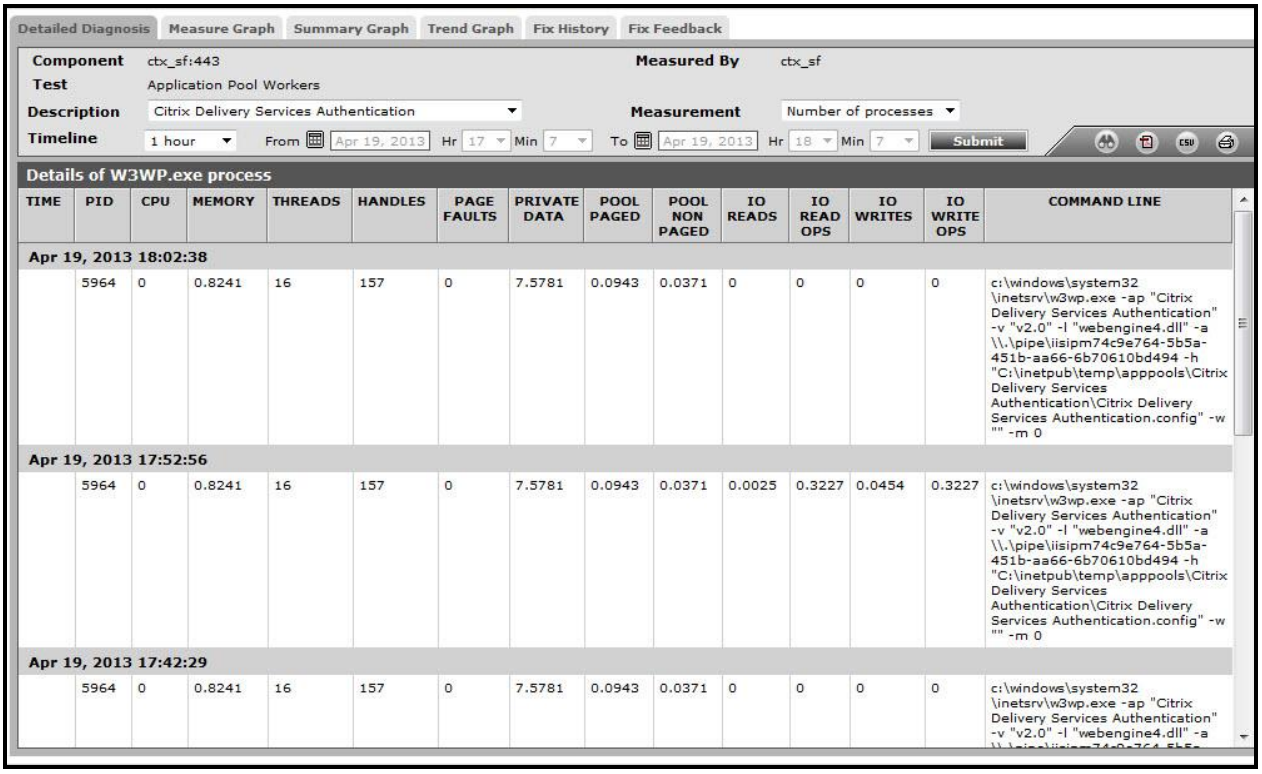
	Startup failure Indicates the number of times that Windows Process Activation Service (WAS) failed to start a worker process that serves requests to this application pool.	Number	Ideally, the value of this measure should be 0. **This measure is reported for IIS v7 (and above) only.**
	Total failure Indicates the number of times the worker processes handling requests to this pool have crashed since the application pool was started.	Number	Ideally, the value of this measure should be 0. **This measure is reported for IIS v7 (and above) only.**
	Ping failure Indicates the number of times the Windows Process Activation Service (WAS) did not receive a response to ping messages sent to a worker process that serves requests to this application pool.	Number	IIS periodically pings a worker process to determine its responsiveness. A high value of this measure indicates that very often the worker process is too busy to respond to these ping requests. This may sometimes force IIS to terminate the worker process. Since the worker process clings on to areas of memory, the termination attempt may fail. As a result, errors/delays may occur when users attempt to hit specific web pages on the IIS web server. To avoid this, you can either disable the Ping, or increase the Ping Maximum Response Time. **This measure is reported for IIS v7 (and above) only.**
	Shutdown failure Indicates the number of times that Windows Process Activation Service (WAS) failed to shut down a worker process servicing requests to this application pool.	Number	**This measure is reported for IIS v7 (and above) only.**
	Initialized threads: Indicates the number of threads that are currently in an initialized state in this application pool.	Number	
	Ready threads: Indicates the number of threads that are currently in the 'Ready' state in this application pool.	Number	A **Ready** thread wants to use a processor but is waiting because none is free. A high value for this measure is therefore indicative of a potential processor contention, probably caused by an overload or a processing bottleneck in the application pool.

	Running threads: Indicates the numberof threads that are currently in the 'Running' state in this application pool.	Number	This measure is a good indicator of the current workload on the application pool.
	Threads in STANDBY state: Indicates the number of threads that are currently in the STANDBY state in this application pool.	Number	A **STANDBY** thread is about to use a processor.
	Terminated threads: Indicates the number of threads that are currently in a **TERMINATED** state in this application pool.	Number	A low value is desired for this measure.
	Threads in WAIT state: Indicates the number of threads that are in the **WAIT** state in this application pool.	Number	A waiting thread has no use for a processor because it is waiting for a peripheral operation or a resource to become free. A high value for this measure could hint at a potential resource contention.
	Threads in TRANSITION state: Indicates the number of threads that are currently in the **TRANSITION** state in this pool.	Number	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 a disk.
	Threads in UNKNOWN state: Indicates the number of threads that are currently in the **UNKNOWN** state in this application pool.	Number	Ideally, the value of this measure should be 0.
	Percentage of running threads: Indicates the percentage of threads that are currently in the **RUNNING** state in this application pool.	Percent	A high value is desired for this measure, as it indicates that that the application pool has adequate active threads to process its requests.
	Active requests: Indicates the current number of requests being processed by the worker process servicing requests to this pool.	Number	

	Active threads: Indicates the number of threads actively processing requests in the worker process that is serving requests to this pool.	Number	
	Current file cache memory usage: Indicates the amount of memory used by user-mode file cache of the worker process that is serving requests to this pool.	MB	
	Current files cached: Indicates the current number of files whose contents are present in the user-mode file cache of the worker process that is serving requests to this pool.	Number	
	Requests rate: Indicates the number of HTTP requests/sec being processed by the worker process that is serving requests to this pool.	Requests/Sec	
	Total requests served: Indicates the total number of HTTP requests served by the worker process that is serving requests to this pool..	Number	
	Total threads: Indicates the total number of threads available to process requests in the worker process that is serving requests to this pool.	Number	By closely monitoring the thread usage over time, you can proactively capture when the worker process runs out of idle threads, and promptly take corrective measures, so as to avert any processing bottlenecks.
	Max threads from thread pool: Indicates the maximum number of threads to which this thread pool can grow.	Number	

Use the detailed diagnosis of the *Number of processes* measure to know the **PID** of every worker process that is handling requests to this pool and the command line argument revealing the name of the application/web site in the application pool, which is accessed by that worker proces. In addition, the detailed diagnosis also displays the

resource allocations (private data, pool paged data, non pool paged data), the resource usage (CPU and memory), open handle count, thread count, and I/O activity of each application in the pool that is accessed by every worker process. This way, resource-intensive applications in the pool can be isolated.



TIME	PID	CPU	MEMORY	THREADS	HANDLES	PAGE FAULTS	PRIVATE DATA	POOL PAGED	POOL NON PAGED	IO READS	IO READ OPS	IO WRITES	IO WRITE OPS	COMMAND LINE
Apr 19, 2013 18:02:38														
	5964	0	0.8241	16	157	0	7.5781	0.0943	0.0371	0	0	0	0	c:\windows\system32 \inetsrv\w3wp.exe -ap "Citrix Delivery Services Authentication" -v "v2.0" -l "webengine4.dll" -a \\.\pipe\iisipm74c9e764-5b5a-451b-aa66-6b70610bd494 -h "C:\inetpub\temp\apppools\Citrix Delivery Services Authentication\Citrix Delivery Services Authentication.config" -w "" -m 0
Apr 19, 2013 17:52:56														
	5964	0	0.8241	16	157	0	7.5781	0.0943	0.0371	0.0025	0.3227	0.0454	0.3227	c:\windows\system32 \inetsrv\w3wp.exe -ap "Citrix Delivery Services Authentication" -v "v2.0" -l "webengine4.dll" -a \\.\pipe\iisipm74c9e764-5b5a-451b-aa66-6b70610bd494 -h "C:\inetpub\temp\apppools\Citrix Delivery Services Authentication\Citrix Delivery Services Authentication.config" -w "" -m 0
Apr 19, 2013 17:42:29														
	5964	0	0.8241	16	157	0	7.5781	0.0943	0.0371	0	0	0	0	c:\windows\system32 \inetsrv\w3wp.exe -ap "Citrix Delivery Services Authentication" -v "v2.0" -l "webengine4.dll" -a

Figure 3.9: The detailed diagnosis of the Number of processes measure of the Application Worker Processes test

3.3.5 IIS Application Pools Test

x. An Application Pool can contain one or more applications and provides a level of isolation between different Web applications. For example, if you want to isolate all the Web applications running in the same computer, you can do this by creating a separate application pool for every Web application and placing them in their corresponding application pool. Because each application pool runs in its own worker process, errors in one application pool will not affect the applications running in other application pools.

y. The IIS Application Pools test discovers all the application pools configured on an IIS Web server and monitors their status. **Note that this test will work on IIS v. 6.0 and above only.**

Purpose	Discovers all the application pools configured on an IIS Web server and monitors their status
Target of the test	IIS web server
Agent deploying the test	An internal agent

Configurable parameters for the test	1. **TEST PERIOD** - How often should the test be executed 2. **HOST -** The host for which the test is to be configured 3. **PORT -** The port to which the specified **HOST** listens 4. **DETAILED DIAGNOSIS** - To make diagnosis more efficient and accurate, the eG Enterprise suite embeds an optional detailed diagnostic capability. With this capability, the eG agents can be configured to run detailed, more elaborate tests as and when specific problems are detected. To enable the detailed diagnosis capability of this test for a particular server, choose the **On** option. To disable the capability, click on the **Off** option. The option to selectively enabled/disable the detailed diagnosis capability will be available only if the following conditions are fulfilled: ➢ The eG manager license should allow the detailed diagnosis capability ➢ Both the normal and abnormal frequencies configured for the detailed diagnosis measures should not be 0.		
Outputs of the test	One set of results for every web site monitored		
Measurements made by the test	**Measurement**	**Measurement Unit**	**Interpretation**
	Application pool status: Indicates the current status of this application pool.	Boolean	If this measure returns the value 1, it indicates that the application pool has started or is starting. While the value 0 for this measure indicates that the application pool has stopped or is stopping, the value -1 indicates that the pool is in an 'Unknown' state. If this measure reports the value 0 or -1, then you need to make sure that the application pool is started. If a pool has stopped, then you can use the detailed diagnosis of this measure to identify the applications that are impacted as a result of this pool stopping.

3.3.6 IIS Web Server Test

This internal test complements the measurements made from an external perspective by the Http test. It collects various statistics that measure the request load on the web server and its responsiveness to the load. The statistics collected by this test are detailed below:

Purpose	To measure the state of a web server
Target of the test	A web server instance
Agent deploying the test	An internal agent

Configurable parameters for the test	1. **TEST PERIOD** - How often should the test be executed 2. **HOST -** The host for which the test is to be configured 3. **PORT -** the port to which the specified **HOST** listens		
Outputs of the test	One set of results for every web server monitored		
Measurements made by the test	**Measurement**	**Measurement Unit**	**Interpretation**
	Connections: Rate of connections to the web server.	Conns/Sec	An increase or decrease in connection rate can represent a change in user workload
	Requests: Rate of requests to the web server during the last measurement period.	Reqs/Sec	With the advent of HTTP/1.1, multiple requests can be transmitted over the same TCP connection. The ratio of requests per connection can provide an idea of the effectiveness of the HTTP 1.1 protocol.
	Data transmitted: Rate at which the data was transmitted by the server during the last measurement period.	KB/Sec	A large increase in the data transmission rate can be indicative of an increase in the popularity of one or more web sites hosted on the server.
	Data received: Rate at which the data was received by the server during the last measurement period.	KB/Sec	An increase in this value is indicative of an increase in user requests to the server.
	Errors: Percentage of error responses from the server during the last measurement period.	Percent	This is the percentage of errors that occurred due to requests that couldn't be satisfied by the server because the requested document could not be found.
	400 errors: Percentage of responses with a status code in the range 400-499 during the last measurement period.	Percent	A high value indicates a number of missing/error pages.
	Current requests: Number of server threads/processes currently in use for serving requests (this measurement is not available for Apache web servers).	Number	If a majority of the server threads/processes are in use simultaneously, this may be indicative of a server bottleneck.

Note:

The Web Server test will report metrics for an IIS web server executing on a Windows 2008 host, only if the **IIS Management Scripts and Tools** feature is installed on that host, and the **Web Server** role you create enables this feature. If the aforesaid feature is not enabled for the **Web Server** role, then remove the role and re-create it with the feature enabled.

3.3.7 HTTP Errors Test

Some errors that occur in IIS are automatically handled by the HTTP API instead of being passed back to IIS for handling. This behavior occurs because the frequency of such errors might otherwise flood an event log or an application handler. The errors captured by the HTTP API are logged in *httperr.log* files that are by default created in the **%SystemRoot%\System32\LogFiles\HTTPERR** folder on the IIS host. These log files provide a wealth of information on the following:

- **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 closes 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 stops unexpectedly while there are still queued requests that are routed to that process. The HTTP API logs the orphaned requests in the error log.

Moreover, specific error types are also designated by **Reason Phrase** strings that always appear as the last field of each error line in the log file.

By periodically parsing these log files and reading the errors/reasons captured by the same, administrators can significantly shorten troubleshooting cycles and rapidly resolve problems. This is where the **HTTP Errors** test helps. This test, at configured intervals, parses the HTTP error log file, reads the reason phrases logged therein, and reports the number of errors that occurred due to each reason. Detailed diagnostics provided by the test also shed more light on the nature of these errors. This way, the test simplifies problem identification, analysis, and resolution.

Purpose	Parses the HTTP error log file, reads the reason phrases logged therein, and reports the number of errors that occurred due to each reason. Detailed diagnostics provided by the test also shed more light on the nature of these errors.
Target of the test	An IIS web server
Agent deploying the test	An internal agent

Configurable parameters for the test	1. **TEST PERIOD** - How often should the test be executed 2. **HOST -** The host for which the test is to be configured 3. **PORT -** the port to which the specified **HOST** listens 4. **LOG FILE PATH** – The *httperr.log* files are by default created in the **%SystemRoot%\System32\LogFiles\HTTPERR** folder on the target server. Since this test, by defaut, looks for the *httperr.log* files in the aforesaid directory only, the **LOG FILE PATH** parameter is set to *none* by default. However, if you have configured these log files to be created in a different directory in your environment, then you will have to explicitly specify the full path to these log files in the **LOG FILE PATH** text box. For instance, your path specification can be: *C:|LogFiles|HttpErrors* 5. **DD FREQUENCY** - Refers to the frequency with which detailed diagnosis measures are to be generated for this test. The default is *1:1*. This indicates that, by default, detailed measures will be generated every time this test runs, and also every time the test detects a problem. You can modify this frequency, if you so desire. Also, if you intend to disable the detailed diagnosis capability for this test, you can do so by specifying *none* against **DD FREQUENCY**. 6. **DETAILED DIAGNOSIS** - To make diagnosis more efficient and accurate, the eG Enterprise suite embeds an optional detailed diagnostic capability. With this capability, the eG agents can be configured to run detailed, more elaborate tests as and when specific problems are detected. To enable the detailed diagnosis capability of this test for a particular server, choose the **On** option. To disable the capability, click on the **Off** option. The option to selectively enabled/disable the detailed diagnosis capability will be available only if the following conditions are fulfilled: • The eG manager license should allow the detailed diagnosis capability • Both the normal and abnormal frequencies configured for the detailed diagnosis measures should not be 0.		
Outputs of the test	One set of results for the IIS web server monitored		
Measurements made by the	**Measurement**	**Measurement Unit**	**Interpretation**

test	**Application offline:** Indicates the number of errors logged in the log file with *AppOffline* as the *Reason Phrase*.	Number	If the **Reason Phrase** is *AppOffline*, it implies that a service unavailable error occurred (an HTTP error 503). The service is not available because application errors caused the application to be taken offline. The detailed diagnosis of this measure reveals the complete details of errors with *AppOffline* as the *Reason Phrase*. Such details include, the date and time of the error, the IP address and port of the affected client, the IP address and port of the affected server, the version of the protocol being used, the verb state that the last request that is parsed passes, the URL requested and any query that is associated with it, the protocol status of the response to the request, the reason phrase, and the request queue name. From these details, you can instantly figure out when the error occurred, the client and server affected by the error, the request in response to which the error occurred, and the reason for the error.
	Busy application pool processes: Indicates the number of errors logged in the log file with *AppPoolTimer* as the *Reason Phrase*.	Number	If the **Reason Phrase** is *AppPoolTimer*, it implies that a service unavailable error occurred (an HTTP error 503), because the application pool process was too busy to handle the request. The detailed diagnosis of this measure reveals the complete details of errors with *AppPoolTimer* as the *Reason Phrase*. Such details include, the date and time of the error, the IP address and port of the affected client, the IP address and port of the affected server, the version of the protocol being used, the verb state that the last request that is parsed passes, the URL requested and any query that is associated with it, the protocol status of the response to the request, the reason phrase, and the request queue name. From these details, you can instantly figure out when the error occurred, the client and server affected by the error, the request in response to which the error occurred, and the reason for the error.

	Application shutdown: Indicates the number of errors logged in the log file with *AppShutdown* as the *Reason Phrase*.	Number	If the **Reason Phrase** is *AppShutdown*, it implies that a service unavailable error occurred (an HTTP error 503), because application shut down automatically in response to administrator policy. The detailed diagnosis of this measure reveals the complete details of errors with *AppShutdown* as the *Reason Phrase*. Such details include, the date and time of the error, the IP address and port of the affected client, the IP address and port of the affected server, the version of the protocol being used, the verb state that the last request that is parsed passes, the URL requested and any query that is associated with it, the protocol status of the response to the request, the reason phrase, and the request queue name. From these details, you can instantly figure out when the error occurred, the client and server affected by the error, the request in response to which the error occurred, and the reason for the error.
	Bad requests: Indicates the number of errors logged in the log file with *BadRequest* as the *Reason Phrase*.	Number	If the **Reason Phrase** is *BadRequest*, it implies that a parse error occurred while processing a request. The detailed diagnosis of this measure reveals the complete details of errors with *BadRequest* as the *Reason Phrase*. Such details include, the date and time of the error, the IP address and port of the affected client, the IP address and port of the affected server, the version of the protocol being used, the verb state that the last request that is parsed passes, the URL requested and any query that is associated with it, the protocol status of the response to the request, the reason phrase, and the request queue name. From these details, you can instantly figure out when the error occurred, the client and server affected by the error, the request in response to which the error occurred, and the reason for the error.

	Connections reset by client: Indicates the number of errors logged in the log file with *Client_Reset* as the *Reason Phrase*.	Number	If the **Reason Phrase** is *Client_Reset,* it implies that the connection between the client and the server was closed before the request could be assigned to a worker process. The most common cause of this behavior is that the client prematurely closes its connection to the server. The detailed diagnosis of this measure reveals the complete details of errors with *Client_Reset* as the *Reason Phrase*. Such details include, the date and time of the error, the IP address and port of the affected client, the IP address and port of the affected server, the version of the protocol being used, the verb state that the last request that is parsed passes, the URL requested and any query that is associated with it, the protocol status of the response to the request, the reason phrase, and the request queue name. From these details, you can instantly figure out when the error occurred, the client and server affected by the error, the request in response to which the error occurred, and the reason for the error.
	Connections abandoned by application pool: Indicates the number of errors logged in the log file with *Connection_Abandoned_By_AppPool* as the *Reason Phrase*.	Number	If the **Reason Phrase** is *Connection_Abandoned_By_AppPool*, it implies that a worker process from the application pool has quit unexpectedly or orphaned a pending request by closing its handle. The detailed diagnosis of this measure reveals the complete details of errors with *Connection_Abandoned_By_AppPool* as the *Reason Phrase*. Such details include, the date and time of the error, the IP address and port of the affected client, the IP address and port of the affected server, the version of the protocol being used, the verb state that the last request that is parsed passes, the URL requested and any query that is associated with it, the protocol status of the response to the request, the reason phrase, and the request queue name. From these details, you can instantly figure out when the error occurred, the client and server affected by the error, the request in response to which the error occurred, and the reason for the error.

	Connections abandoned by request queue: Indicates the number of errors logged in the log file with *Connection_Abandoned_By_ReqQueue* as the *Reason Phrase*.	Number	If the **Reason Phrase** is *Connection_Abandoned_By_ReqQueue*, it implies that a worker process from the application pool has quit unexpectedly or orphaned a pending request by closing its handle. **This is specific to Windows Vista and later versions and to Windows Server 2008 and later versions.** The detailed diagnosis of this measure reveals the complete details of errors with *Connection_Abandoned_By_ReqQueue* as the *Reason Phrase*. Such details include, the date and time of the error, the IP address and port of the affected client, the IP address and port of the affected server, the version of the protocol being used, the verb state that the last request that is parsed passes, the URL requested and any query that is associated with it, the protocol status of the response to the request, the reason phrase, and the request queue name. From these details, you can instantly figure out when the error occurred, the client and server affected by the error, the request in response to which the error occurred, and the reason for the error.
	Connections dropped: Indicates the number of errors logged in the log file with *Connection_Dropped* as the *Reason Phrase*.	Number	If the **Reason Phrase** is *Connection_Dropped*, it implies that 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. The detailed diagnosis of this measure reveals the complete details of errors with *Connection_Dropped* as the *Reason Phrase*. Such details include, the date and time of the error, the IP address and port of the affected client, the IP address and port of the affected server, the version of the protocol being used, the verb state that the last request that is parsed passes, the URL requested and any query that is associated with it, the protocol status of the response to the request, the reason phrase, and the request queue name. From these details, you can instantly figure out when the error occurred, the client and server affected by the error, the request in response to which the error occurred, and the reason for the error.

	Connections limit has reached: Indicates the number of errors logged in the log file with *ConnLimit* as the *Reason Phrase*.	Number	If the **Reason Phrase** is *ConnLimit*, it implies that a a service unavailable error occurred (an HTTP error 503), because the site level connection limit has been reached or exceeded. The detailed diagnosis of this measure reveals the complete details of errors with *ConnLimit* as the *Reason Phrase*. Such details include, the date and time of the error, the IP address and port of the affected client, the IP address and port of the affected server, the version of the protocol being used, the verb state that the last request that is parsed passes, the URL requested and any query that is associated with it, the protocol status of the response to the request, the reason phrase, and the request queue name. From these details, you can instantly figure out when the error occurred, the client and server affected by the error, the request in response to which the error occurred, and the reason for the error.
	Connections refused: Indicates the number of errors logged in the log file with *Connections_Refused* as the *Reason Phrase*.	Number	If the **Reason Phrase** is *Connections_Refused*, it implies that the kernel NonPagedPool memory has dropped below 20MB and http.sys has stopped receiving new connections. The detailed diagnosis of this measure reveals the complete details of errors with *Connections_Refused* as the *Reason Phrase*. Such details include, the date and time of the error, the IP address and port of the affected client, the IP address and port of the affected server, the version of the protocol being used, the verb state that the last request that is parsed passes, the URL requested and any query that is associated with it, the protocol status of the response to the request, the reason phrase, and the request queue name. From these details, you can instantly figure out when the error occurred, the client and server affected by the error, the request in response to which the error occurred, and the reason for the error.

	Internal errors: Indicates the number of errors logged in the log file with *Internal* as the *Reason Phrase*.	Number	If the **Reason Phrase** is *Internal*, it implies that an internal server error occurred (an HTTP error 500). The detailed diagnosis of this measure reveals the complete details of errors with *Internal* as the *Reason Phrase*. Such details include, the date and time of the error, the IP address and port of the affected client, the IP address and port of the affected server, the version of the protocol being used, the verb state that the last request that is parsed passes, the URL requested and any query that is associated with it, the protocol status of the response to the request, the reason phrase, and the request queue name. From these details, you can instantly figure out when the error occurred, the client and server affected by the error, the request in response to which the error occurred, and the reason for the error.
	Application request limit reached: Indicates the number of errors logged in the log file with *QueueFull* as the *Reason Phrase*.	Number	If the **Reason Phrase** is *QueueFull*, it implies that a service unavailable error occurred (an HTTP error 503), because the application request queue is full. The detailed diagnosis of this measure reveals the complete details of errors with *QueueFull* as the *Reason Phrase*. Such details include, the date and time of the error, the IP address and port of the affected client, the IP address and port of the affected server, the version of the protocol being used, the verb state that the last request that is parsed passes, the URL requested and any query that is associated with it, the protocol status of the response to the request, the reason phrase, and the request queue name. From these details, you can instantly figure out when the error occurred, the client and server affected by the error, the request in response to which the error occurred, and the reason for the error.

	Connections expired: Indicates the number of errors logged in the log file with *Connection_Dropped_List_Full* as the *Reason Phrase*.	Number	If the **Reason Phrase** is *Connection_Dropped_List_Full*, it implies that the list of dropped connections between clients and the server is full. **This is specific to Windows Vista and later versions and to Windows Server 2008 and later versions.** The detailed diagnosis of this measure reveals the complete details of errors with *Connection_Dropped_List_Full* as the *Reason Phrase*. Such details include, the date and time of the error, the IP address and port of the affected client, the IP address and port of the affected server, the version of the protocol being used, the verb state that the last request that is parsed passes, the URL requested and any query that is associated with it, the protocol status of the response to the request, the reason phrase, and the request queue name. From these details, you can instantly figure out when the error occurred, the client and server affected by the error, the request in response to which the error occurred, and the reason for the error.
	URL processing errors: Indicates the number of errors logged in the log file with *URL* as the *Reason Phrase*.	Number	If the **Reason Phrase** is *URL*, it implies that a parse error occurred while processing a URL. The detailed diagnosis of this measure reveals the complete details of errors with *URL* as the *Reason Phrase*. Such details include, the date and time of the error, the IP address and port of the affected client, the IP address and port of the affected server, the version of the protocol being used, the verb state that the last request that is parsed passes, the URL requested and any query that is associated with it, the protocol status of the response to the request, the reason phrase, and the request queue name. From these details, you can instantly figure out when the error occurred, the client and server affected by the error, the request in response to which the error occurred, and the reason for the error.

3.3.8 HTTP Service Request Queues Test

By monitoring request queues to every IIS application pool, administrators can identify those application pools that have too many requests pending and those with a high request rejection rate. This is exactly what the **HTTP Service Request Queues** test does. This test auto-discovers the application pools and reports the length of request queues

and rejection rate of requests in queue for each application pool. This way, the test sheds light on those application pools that suffer from processing pains.

Purpose	Auto-discovers the application pools and reports the length of request queues and rejection rate of requests in queue for each application pool. This way, the test sheds light on those application pools that suffer from processing pains		
Target of the test	IIS web server		
Agent deploying the test	An internal agent		
Configurable parameters for the test	1. **TEST PERIOD** - How often should the test be executed 2. **HOST -** The host for which the test is to be configured. 3. **PORT** – The port at which the **HOST** listens.		
Outputs of the test	One set of results for every application pool on Exchange		
Measurements made by the test	**Measurement**	**Measurement Unit**	**Interpretation**
	Requests arrival rate: Indicates the rate at which requests are arriving in the request queue of this application pool.	Reqs/Sec	A high rate could indicate a probable request overload on an application pool.
	Requests in queue: Indicates the current number of requests in the queue of this application pool.	Number	If this value increases consistently, it could indicate that the application pool is not processing requests quickly. Compare the value of this measure across pools to identify the pool with a processing bottleneck.
	Requests rejected from queue: Indicates the number of requests in this application pool's queue that were rejected.	Number	A non-zero value is desired for this measure.
	Cache hit rate: Indicates the rate of cache hits from the queue of this application pool.	Hits/sec	
	Request rejection rate: Indicates the rate at which this application pool rejected requests in the queue.	Reqs/Sec	Compare the value of this measure across pools to know which pool rejects queued requests frequently.

	Maximum queue item age: Indicates the age of the oldest request in the queue.	Reqs/Sec	

3.3.9 Tests Disabled by Default

While the tests discussed above are available by default for an IIS web server, eG Enterprise maps a few other tests to the **Web Server** layer, which are disabled by default, but can be optionally enabled by the user. To enable one/more of these tests, select the check box against the test name in the **DISABLED TESTS** list of the **AGENTS – TESTS CONFIGURATION** page (Agents -> Tests -> Configure menu sequence), and click the **Update** button in the page.

3.3.9.1 IIS Web Transactions Test

Whenever users to your mission-critical web sites (on an IIS web server) or web services (overlying an IIS web server) complain of slowdowns, the first step to troubleshooting these issues is identifying which specific transaction(s) to the web sites is causing the slowdown. For this, you will have to periodically monitor the responsiveness of key transactions to your web sites, so as to quickly and accurately identify slow transactions. The **Slow Transactions** test helps you achieve this. This test monitors user-configured transaction patterns, captures the response time of each configured pattern in real-time, compares these actuals with the desired levels of responsiveness (which is also configurable), and thus isolates and proactively alerts you to slow transactions.

This test executes only on IIS web servers operating on Windows 2008.

Note:

To make sure that this test reports measures, do the following:

- Configure a **Web Server** role on the target Windows 2008 server.
- Install and configure **Advanced Logging** for the IIS web server operating on Windows 2008.

Purpose	Monitors user-configured transaction patterns, captures the response time of each configured pattern in real-time, compares these actuals with the desired levels of responsiveness (which is also configurable), and thus isolates and proactively alerts you to slow transactions
Target of the test	An IIS web server
Agent deploying the test	An internal agent

<table>
<tr><td>Configurable parameters for the test</td><td colspan="3">1. TEST PERIOD - How often should the test be executed
2. HOST - The host for which the test is to be configured
4. PORT - The port to which the specified HOST listens
5. NAME, PATTERN, THRESHOLD VALUE - Using these text boxes, you can configure the URL patterns for the transactions to be monitored by this test. You can group a set of related/similar patterns under a common head - a display name for this group head can be provided in the NAME text box; this display name will appear as a descriptor of the test in the eG monitoring console. In the PATTERN text box, you can configure the transaction patterns that are to be grouped under the specified NAME. Multiple transaction patterns can be separated by a semi-colon. Wild card characters can be used while configuring these patterns. For example, to monitor all transactions with the extension html or jsp, your PATTERN specification should be: *.html;*.jsp. In the THRESHOLD VALUE text box, specify the response time value (in seconds), which should be violated, for a transaction to be counted as a 'slow transaction'. All transaction patterns grouped under a given NAME will be governed by the THRESHOLD VALUE specified.
You can, if you so need, configure multiple display NAMEs, PATTERNs, and THRESHOLD VALUEs for monitoring. To enable you to configure the multiple values easily, the eG administrative interface provides a special page. To access this page, click on the Click here hyperlink, which appears just above the parameters of this test in the TEST CONFIGURATION PAGE. Refer to Section 3.3.9.1.1 of this document to know how to use this special page.
6. DETAILED DIAGNOSIS - To make diagnosis more efficient and accurate, the eG Enterprise suite embeds an optional detailed diagnostic capability. With this capability, the eG agents can be configured to run detailed, more elaborate tests as and when specific problems are detected. To enable the detailed diagnosis capability of this test for a particular server, choose the On option. To disable the capability, click on the Off option.
The option to selectively enable/disable the detailed diagnosis capability will be available only if the following conditions are fulfilled:
• The eG manager license should allow the detailed diagnosis capability
• Both the normal and abnormal frequencies configured for the detailed diagnosis measures should not be 0.</td></tr>
<tr><td>Outputs of the test</td><td colspan="3">One set of results for every NAME configured</td></tr>
<tr><td rowspan="3">Measurements made by the test</td><td>Measurement</td><td>Measurement Unit</td><td>Interpretation</td></tr>
<tr><td>Request rate:
Indicates the rate of requests to this transaction.</td><td>Requests/Sec</td><td></td></tr>
<tr><td>Requests processed:
Indicates the number of requests to this transaction during the last measurement period.</td><td>Requests</td><td></td></tr>
</table>

	Successful responses: Indicates the percentage of responses with the response code of 200.	Percent	Typically, successful requests are indicated using the response code 200 - 299.
	Redirects: Indicates the percentage of responses with the response code of 300.	Percent	300 responses could indicate page caching on the client browsers. Alternatively 300 responses could also indicate redirection of requests. A sudden change in this value could indicate a problem condition.
	Responses with Status 400: Indicates the percentage of responses with the response code of 400.	Percent	A high value indicates a number of missing/error pages.
	Responses with Status 500: Indicates the percentage of responses with the response code of 500.	Percent	Since responses with a status code of 500-600 indicate server side processing errors, a high value reflects an error condition.
	Min response time: Indicates the minimum response time of this transaction.	Secs	
	Max response time: Indicates the maximum response time of this transaction.	Secs	
	Avg. response time: Indicates the average response time for all requests to this transaction.	Secs	A low value is desired for this measure. A sudden or steady increase in this value could indicate a problem with the transaction.
	Data transmitted: Indicates the total data transmitted by this transaction.	KB	
	Avg. data transmitted: Indicates the data transmitted by this transaction on an average.	KB	

	Slow responses: Indicates the number of requests to this transaction with a response time greater than the specified **THRESHOLD VALUE**.	Number	Ideally, the value of this measure should be 0. A non-zero value indicates the existence of one/more slow transactions. Comparing the value of this measure across transactions will quickly point you to the transactions that are very slow. Using the detailed diagnosis of this measure you can view the complete details of the slow transactions.
	Slow response ratio: Indicates the percentage of requests to this transaction that were serviced slowly - i.e., the percentage of requests with a response time greater than the specified **THRESHOLD VALUE**.	Percent	A very low value is desired for this measure. When users complaint of slowdowns while trying to access a web site (on the monitored IIS web server), you may want to compare the value of this measure across transactions to know which transaction is the slowest, and hence could be causing the slowdown.
	Avg. response time for slow responses: Indicates the average response time of the slow requests to this transaction.	Secs	
	Slow response threshold: Indicates the **THRESHOLD VALUE** configured for this transaction.	Secs	

Using the detailed diagnosis of the **Number of slow requests** measure you can know when the slow transactions occurred, identify the users who initiated the slow transactions, the IP address from which the transaction requests were received, and the URL that was accessed to execute the transaction. The response time registered by the transaction whenever it occurred will also be displayed, along with the data transmitted and received. With the help of this information, you can analyze why the transactions experienced slowdowns and initiate corrective action.

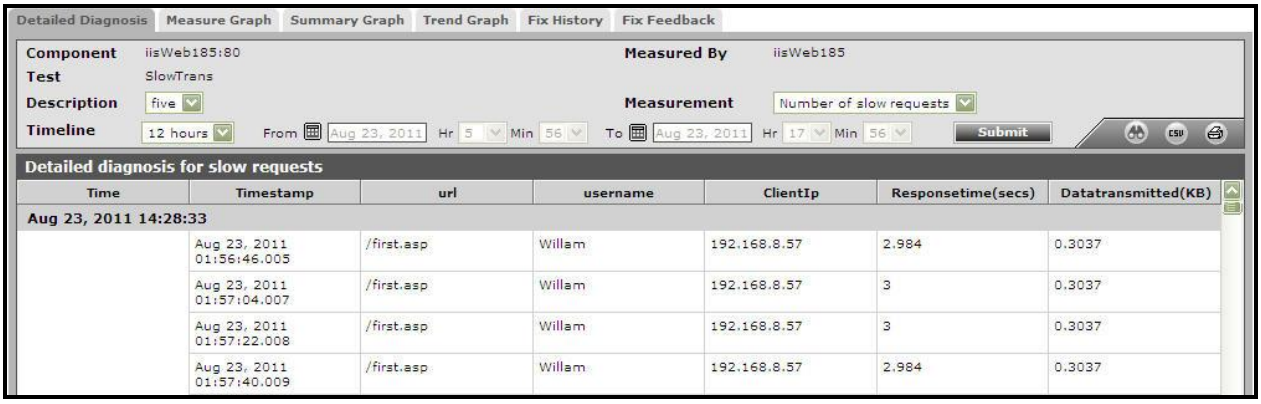


Figure 3.10: The detailed diagnosis of the Number of slow requests measure

Note:

Sometimes, the **Advanced Logging** module of IIS may not be able to log the username information related to web transactions. In such cases, no values will be displayed in the **username** column of the detailed diagnosis of the **IIS Web Transactions** test (see Figure 3.9). The workaround for this issue is provided below:

1. Login to the IIS host.
2. Open the **C:\Windows\System32\inetsrv\config\applicationHost.config** file in a text editor.
3. Locate the following line in the file:

```
<field id="UserName" sourceName="UserName" sourceType="RequestHeader"
logHeaderName="cs-username" category="Default" loggingDataType="TypeLPCSTR" />
```

4. Change the **sourceType** value to "BuiltIn" instead of "RequestHeader" as shown below:

```
<field id="UserName" sourceName="UserName" sourceType="BuiltIn"
logHeaderName="cs-username" category="Default" loggingDataType="TypeLPCSTR" />
```

5. Save the file.

3.3.9.1.1 Configuring Multiple URL Patterns for Monitoring

To enable administrators to easily configure the **Slow Transactions** test with multiple **NAME**s, **PATTERN**s, and **THRESHOLD VALUE**s, the eG administrative interface offers a special page. To access this page, you need to click on the **Click here** hyperlink, which will be available just about the parameters of this test in the **TEST CONFIGURATION** page (see Figure 3.11).

Figure 3.11: Configuring the Slow Transactions test

Doing so will invoke Figure 3.12, using which multiple URL patterns can be configured

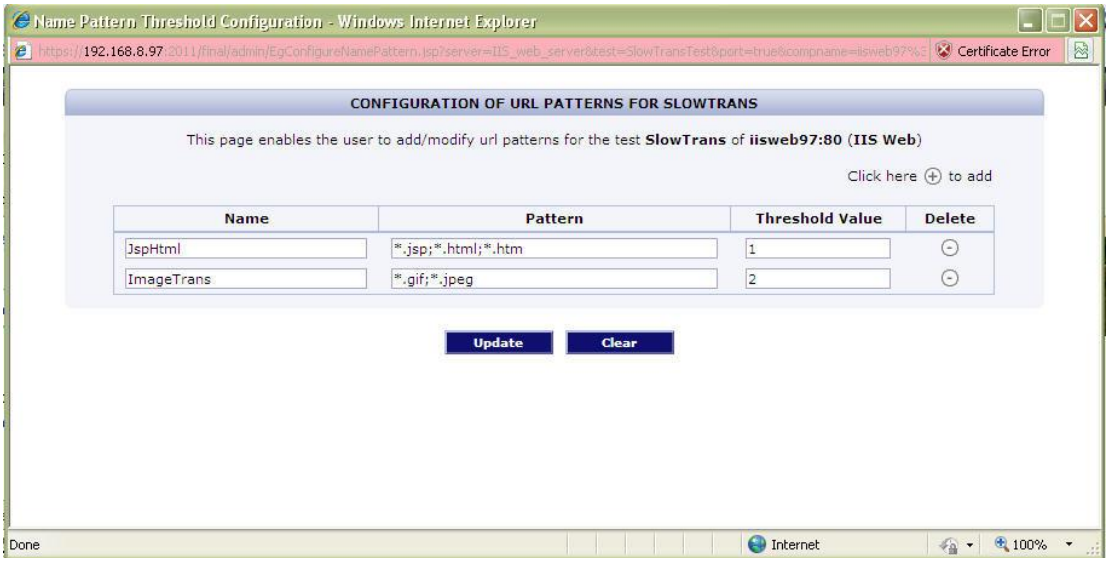


Figure 3.12: Configuring multiple URL patterns

To do so, follow the steps below:

1. Provide a display name for related/similar URL patterns in the **NAME** text box. This display name will appear as a descriptor of the test in the eG monitoring console.
2. In the **PATTERN** text box, you can configure the transaction patterns that are to be grouped under the specified **NAME**. Multiple transaction patterns can be separated by a semi-colon. Wild card characters can be used while configuring these patterns. For example, to monitor all transactions with the extension **html** or **jsp**, your **PATTERN** specification should be: **.html;*.jsp*.
3. In the **THRESHOLD VALUE** text box, specify the response time value (in seconds), which should be violated, for a transaction to be counted as a 'slow transaction'. All transaction patterns grouped under a given **NAME** will be governed by the **THRESHOLD VALUE** specified.
4. To add another specification, click on the encircled '+' button in Figure 3.12. This will invoke another row,

where you can provide another **NAME**, associate a set of **PATTERN**s with that **NAME**, and configure a **THRESHOLD VALUE** for those patterns.

5. To delete a specification, simply click on the encircled '-' button at the end of every row in Figure 3.12.
6. To clear all your specifications at one shot, click on the **Clear** button in Figure 3.12.
7. To save the changes, click on the **Update** button.
8. You will then return to the **TEST CONFIGURATION PAGE**, where you can view all your specifications (see Figure 3.13).

Figure 3.13: The Slow Transactions test configured ith multiple URL patterns

3.3.9.2 Windows Process Activation Service Test

The Windows Process Activation Service (WAS) manages application pool configuration and the creation and lifetime of worker processes for HTTP and other protocols. The World Wide Web Publishing Service (W3SVC) and other services depend on WAS.

This test monitors the Windows Process Activation Service and reports useful statistics revealing how well the service manages the worker processes and how healthy these worker processes are.

Purpose	Monitors the Windows Process Activation Service and reports useful statistics revealing how well the service manages the worker processes and how healthy these worker processes are		
Target of the test	An IIS web server		
Agent deploying the test	An internal agent		
Configurable parameters for the test	1. **TEST PERIOD** - How often should the test be executed 2. **HOST -** The host for which the test is to be configured 3. **PORT -** the port to which the specified **HOST** listens		
Outputs of the test	One set of results for every worker process on the IIS web server being monitored		
Measurements made by the	**Measurement**	**Measurement Unit**	**Interpretation**

test	**Active listener channels**: Indicates the number of currently active listener channels in the worker process.	Number	
	Active protocol handlers: Indicates the number of currently active protocol handlers in the worker process.	Number	
	Health ping reply latency: Indicates the average time taken by worker process to reply to last health ping.	Millisecs	You can monitor and improve application pool health by having the Windows Process Activation Service (WAS) ping an application pool's worker process at set intervals. A delayed response from the worker process or the lack of any response might mean that the worker process does not have a thread to respond to the ping request, or that it is hanging for some other reason. Based on the results of the ping request, WAS can flag a worker process as unhealthy and shut it down. By default, worker process pinging is enabled. You may have to adjust the ping interval and the ping response time to gain access to timely information about application pool health without triggering false unhealthy conditions, for example, instability caused by an application.
	Total requests served: Indicates the total number of requests served by the worker process.	Number	This counter is only meaningful when request based recycling is enabled for the application pool.

3.3.9.3 Worker Processes Statistics Test

This test exposes how well the worker processes on an IIS web server are handling the HTTP requests to the server. Using the metrics reported you can understand the extent of load handled by the worker processes and can determine whether the worker processes have enough threads for processing requests.

Purpose	Exposes how well the worker processes on an IIS web server are handling the HTTP requests to the server
Target of the test	An IIS web server
Agent deploying the	An internal agent

test			
Configurable parameters for the test	1. **TEST PERIOD** - How often should the test be executed 2. **HOST -** The host for which the test is to be configured 3. **PORT -** the port to which the specified **HOST** listens		
Outputs of the test	One set of results for every worker process on the IIS web server being monitored		
Measurements made by the test	**Measurement**	**Measurement Unit**	**Interpretation**
	Active requests: Indicates the current number of requests being processed by the worker processes.	Number	
	Active threads: Indicates the number of threads actively processing requests in the worker process.	Number	
	Current file cache memory usage: Indicates the amount of memory used by user-mode file cache.	MB	
	Current files cached: Indicates the current number of files whose contents are present in the user-mode file cache.	Number	
	Requests rate: Indicates the number of HTTP requests/sec being processed by the worker process.	Requests/Sec	
	Total requests served: Indicates the total number of HTTP requests served by the worker process.	Number	
	Total threads: Indicates the total number of threads available to process requests in the worker process.	Number	By closely monitoring the thread usage over time, you can proactively capture when the worker process runs out of idle threads, and promptly take corrective measures, so as to avert any processing bottlenecks.

3.3.9.4 ASP Test

The ASP test monitors the performance of an IIS web server for servicing requests to Active Server Pages (ASPs). This test will be disabled by default.

Purpose	To monitor the performance of an IIS web server for servicing requests to Active Server Pages (ASPs)		
Target of the test	An IIS web server		
Agent deploying the test	An internal agent		
Configurable parameters for the test	1. **TEST PERIOD** - How often should the test be executed 2. **HOST -** The host for which the test is to be configured 3. **PORT -** the port to which the specified **HOST** listens		
Outputs of the test	One set of results for every IIS web server being monitored		
Measurements made by the test	**Measurement**	**Measurement Unit**	**Interpretation**
	ASP requests: Indicates the rate of requests for ASP pages during the last measurement period.	Reqs/Sec	
	Transactions for ASP pages: Indicates the rate of transactions started per second	Trans/Sec	
	Error rate: Indicates the rate of errors during ASP processing.	Errors/Sec	
	Data received: Indicates the rate of data being received by the web server for ASP processing,	Kbytes/Sec	
	Data transmitted: Indicates the rate of data being transmitted by the web server in response to ASP requests during the last measurement period.	Kbytes/Sec	

	Execution time: Indicates the execution time of the last request in seconds. To determine where the bottleneck is, compare the execution time with the wait time.	Secs	If the wait time is low, but execution time is high, this indicates that the application logic in the ASP pages could be causing the high execution time. On the other hand, if wait time is high and execution time is low, this indicates a server-side bottleneck (e.g., due to too few processing threads).
	Wait time for ASP execution: Indicates the amount of time that the last request had to wait for service.	Secs	During ideal operation, the wait time should be near 0. If this value is high, it indicates that many requests are being queued for ASP processing.
	Current executions: Indicates the number of requests being currently executed by the ASP engine.	Number	
	Current queue length: Indicates the number of requests that are currently waiting for service.	Number	Ideally, this metric should be 0. A consistent non-zero value is an indicator of a server-side processing bottleneck.
	Failed requests: Indicates the number of requests that failed during the last measurement period. This includes authorization failures and rejections.	Number	
	Rejected requests: Indicates the number of requests that were rejected during the last measurement period.	Number	
	Pending transactions: Indicates the number of transactions awaiting processing during the last measurement period.	Number	
	Transaction commits: Indicates the number of transactions that were committed during the last measurement period.	Number	

	Transaction aborts: Indicates the number of transactions that were aborted during the last measurement period.	Number	A very high value of this measure could indicate a server-processing bottleneck.

3.3.9.5 W3 WP Pools Test

Unlike IIS on Windows 2000, where all web sites operate within a logical default application pool, multiple application pools can be physically configured on IIS web servers on Windows 2003. Typically, each of these application pools will contain a web site. Problems in a web site will therefore, adversely impact the performance of the corresponding application pool, sometimes causing the pool to crash. In order to prevent such adversities, you can use the W3WPPool test to closely observe application pool performance (for IIS web servers on Windows 2003) in terms of resource usage, and promptly report abnormalities (if any). Using the statistics gathered by this test, you not only get to identify the problematic application pool, but also the erroneous web site.

This test is disabled by default, and can be enabled when monitoring an IIS web server on Windows 2003.

Purpose	Measures the resource usage of the application pools on an IIS web server operating on Windows 2003
Target of the test	An IIS web server
Agent deploying the test	An internal agent

<table>
<tr><td>Configurable parameters for the test</td><td colspan="3">1. TEST PERIOD - How often should the test be executed
2. HOST - The host for which the test is to be configured
3. PORT - the port to which the specified HOST listens
4. APPLICATION POOL NAME - Provide a comma-separated list of application pools to be monitored. The default value is "all", indicating that all available application pools will be monitored by default. To know the application pools on the IIS web server in question, do the following:
➢ On the IIS host, follow the menu sequence: Start -> Programs -> Administrative Tools -> Internet Information Services (IIS) Manager.
➢ When the IIS Manager opens, expand the node that corresponds to the local host in the tree-structure in the left pane of the manager.
➢ An Application Pools sub-node will appear. Expand this sub-node to view the complete list of application pools on the IIS web server.
5. DETAILED DIAGNOSIS - To make diagnosis more efficient and accurate, the eG Enterprise suite embeds an optional detailed diagnostic capability. With this capability, the eG agents can be configured to run detailed, more elaborate tests as and when specific problems are detected. To enable the detailed diagnosis capability of this test for a particular server, choose the On option. To disable the capability, click on the Off option.
The option to selectively enable/disable the detailed diagnosis capability will be available only if the following conditions are fulfilled:
➢ The eG manager license should allow the detailed diagnosis capability
➢ Both the normal and abnormal frequencies configured for the detailed diagnosis measures should not be 0.</td></tr>
<tr><td>Outputs of the test</td><td colspan="3">One set of results for every web site monitored</td></tr>
<tr><td>Measurements made by the test</td><td>Measurement</td><td>Measurement Unit</td><td>Interpretation</td></tr>
<tr><td></td><td>Number of processes:
Indicates number of w3wp.exe processes that are currently running in the application pool.</td><td>Number</td><td>A zero value in this measure indicates that the application pool is crashed.</td></tr>
<tr><td></td><td>CPU utilization:
Indicates the percentage CPU utilization of the application pool.</td><td>Percent</td><td>A higher value indicates excessive CPU utilization.</td></tr>
<tr><td></td><td>Memory utilization:
Indicates the percentage of total memory utilized by the application pool.</td><td>Percent</td><td>If the value of this measure consistently increases, it indicates a memory bottleneck.</td></tr>
</table>

	Number of threads: Indicates the number of threads that are currently active in the application pool.	Number	This is an indicator of the workload on the pool.
	Number of handles: Indicates the number of handles currently opened by the application pool.	Number	
	Page faults: Indicates the rate of page faults happening in the application pool.	Faults/Sec	
	Private data: Indicates the amount of data that the application pool has currently allocated and cannot be shared with other application pools.	MB	A gradual growth in this measure indicates a memory leak in the websites that are running in the application pool.
	Pool paged data: Indicates the amount of memory currently allocated from the paged pool. The Paged Pool is an area of the System's virtual memory that is limited in size and used for various system related functions.	MB	A gradual growth in this measure indicates a memory leak in the websites that are running in the application pool.
	Pool non paged data: Indicates the amount of memory currently allocated from the non-paged pool. The Non-Paged Pool is an area of the System's virtual memory that is limited in size and used for kernel and device driver functions.	MB	A gradual growth in this measure indicates a memory leak in the websites that are running in the application pool. Running out of space in the nonpaged pool often results in a Blue Screen.
	I/O reads: Indicates the rate at which the W3WP.exe process is reading data from I/O operations.	KBytes/Sec	

	I/O read operations: Indicates the rate at which the W3WP.exe process is issuing read I/O operations.	Operations/Sec	
	I/O writes: Indicates rate at which the W3WP.exe process is writing data into I/O operations.	KBytes/Sec	
	I/O write operations: Indicates the rate at which the W3WP.exe process is issuing write I/O operations	Operations/Sec	

3.3.9.6 Web Site Traffic Test

The **Web Site** layer executes a WebSiteTest that interfaces with the eG web adapter to collect statistics related to the usage of every web site configured on a web server. One of the key constraints of this test is that three critical usage metrics, namely, **Aborts**, **300 responses**, **500 errors**, will not be available for web servers executing on Windows environments. Therefore, for an IIS web server naturally, the eG agent will not be able to report these measures. In order to ensure that the above-mentioned error-related statistics are available to the IIS web servers also, eG Enterprise offers a separate **Web Site Traffic test**; this test executes only on IIS/IIS SSL web servers to monitor the traffic to and from each of the web sites on these servers, the errors that occurred during web site accesses, etc. This test reads the log files on the monitored IIS/IIS SSL web servers to extract important metrics pertaining to the web site usage. These log files are created only if **logging is explicity enabled** on the monitored IIS/IIS SSL web server. To enable logging, do the following:

1. On the IIS web server host, open the **Internet Information Services (IIS) Manager** using the following sequence of steps: Start -> Settings -> Control Panel -> Administrative Tools -> Internet Information Services (IIS) Manager.

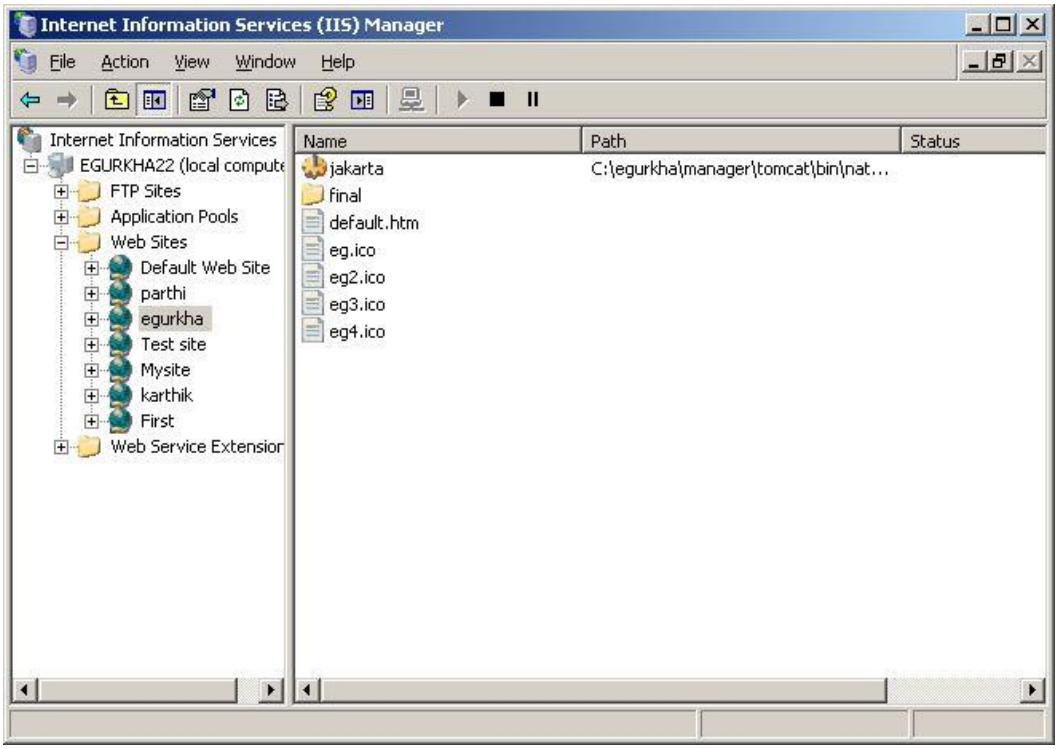


Figure 3.14: The IIS Manager console

2. In the tree-structure in the left pane of the console (see Figure 3.14), expand the node representing the local host, and then, the **Web Sites** node within; the complete list of web sites configured on the IIS web server will appear here.

3. To enable logging for a particular web site, select that web-site from under the **Web Sites** node, right-click on it, and then, select the **Properties** option from the shortcut menu (see Figure 3.15).

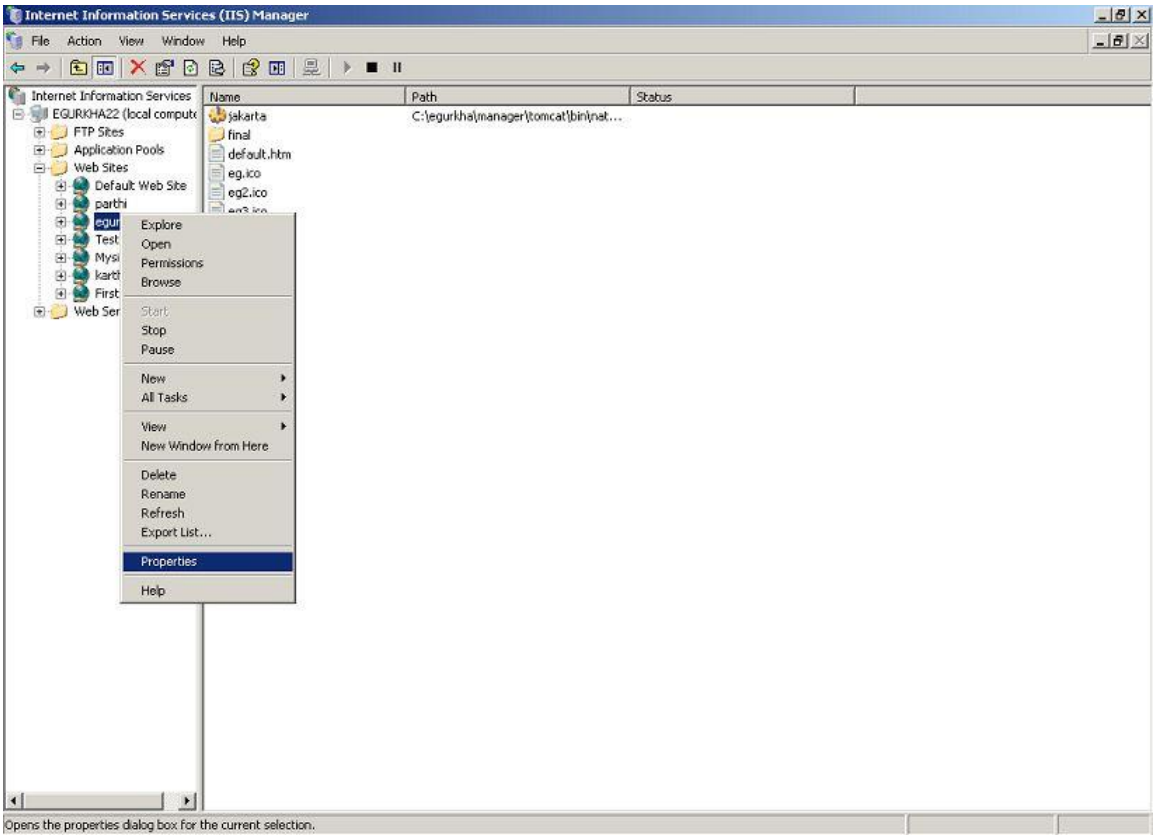


Figure 3.15: Selecting the Properties option of the web site

4. In the **Web Site** tab of the **Properties** dialog box that appears (see Figure 3.16), ensure that the **Enable Logging** check box is selected. Also, make sure that **W3C Extented Log File Format** is chosen as the **Active log format**.

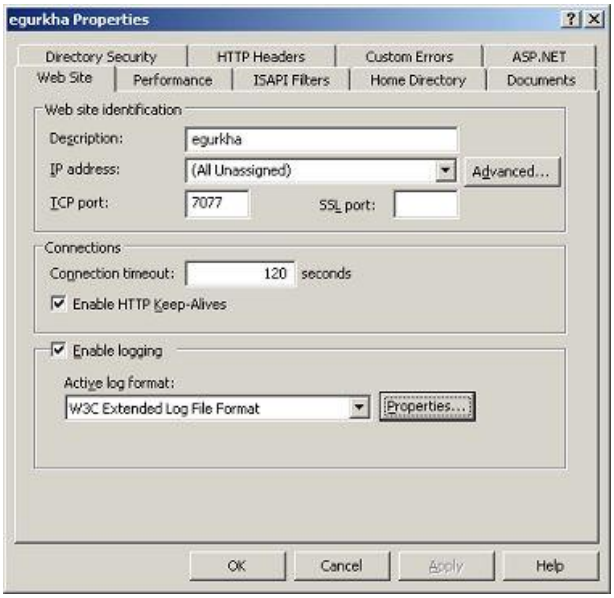


Figure 3.16: The Properties of a web site on a web server

5. Next, click on the **Properties** button adjacent to the **Active log format** chosen. Figure 3.17 then appears, displaying the **General** settings of the **W3C Extented Log File Format**. In the **Log file directory** text box here, specify the directory to which the log file should be written; by default, **C:\Windows\system32\LogFiles** is displayed therein.

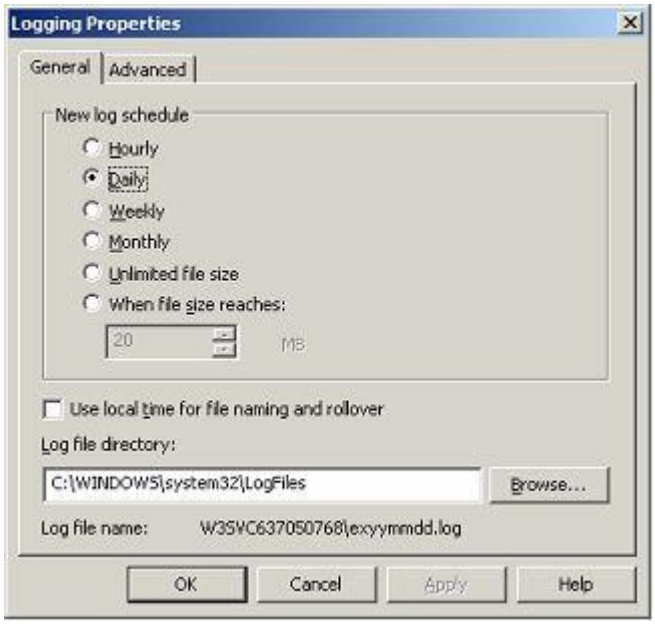


Figure 3.17: The General settings of the Active log format

6. Click on the **Advanced** tab in Figure 3.17 to indicate the type of statistics to be logged into the log file. In Figure 3.18 that appears, ensure that the **Win32 Status**, **Bytes Sent**, and **Bytes Received** check boxes are selected.

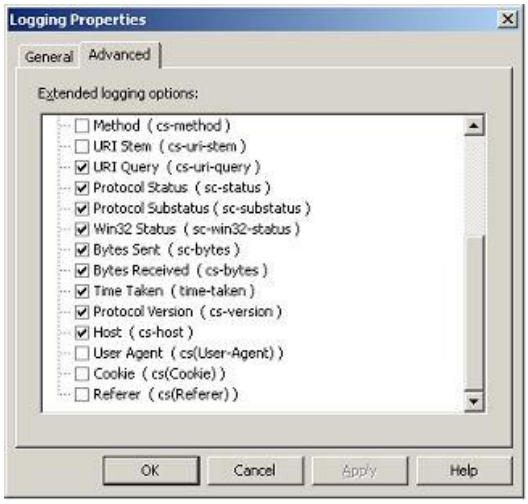


Figure 3.18: Selecting the information to be logged

7. Finally, click the **Apply** and **OK** buttons to register the changes.

8. Repeat this procedure for every web site to be monitored.

This test is disabled by default, and can be enabled while monitoring an IIS/IIS SSL web server.

Purpose	To extract important metrics pertaining to the traffic to and from each of the web sites on the IIS/IIS SSL web servers, the errors that occurred during web site accesses, etc.		
Target of the test	A web server instance		
Agent deploying the test	An internal agent		
Configurable parameters for the test	1. **TEST PERIOD** - How often should the test be executed 2. **HOST -** The host for which the test is to be configured 3. **PORT -** the port to which the specified **HOST** listens 4. **LOGFILEPATH** - Specify the full path to directory on the target IIS/IIS SSL web server that stores the log files; here, you need to specify the same directory that is provided in the **Log file directory** text box of Figure 5.10.		
Outputs of the test	One set of results for every web site configured for monitoring on the target IIS/IIS SSL web server		
Measurements made by the test	**Measurement**	**Measurement Unit**	**Interpretation**
	Requests handled: Indicates the number of requests currently received by this web site.	Number	
	Data transmitted (MB): Indicates the amount of data currently sent by this web site.	MB	A large increase in the data transmission can be indicative of an increase in the popularity of one or more web sites hosted on the server.
	Data received (MB): Indicates the amount of data currently received by this web site.	MB	An increase in this value is indicative of an increase in user requests to the server.
	200 responses: Indicates the percentage of responses with the status code 200-299 during the last measurement period.	Percent	Typically, successful requests are indicated using the response code 200 - 299.
	300 responses: Percentage of responses with a status code in the 300-399 range during the last measurement period.	Percent	300 responses could indicate page caching on the client browsers. Alternatively 300 responses could also indicate redirection of requests. A sudden change in this value could indicate a problem condition.

	400 errors: Percentage of responses with a status code in the range 400-499 during the last measurement period.	Percent	A high value indicates a number of missing/error pages.
	500 errors: Percentage of responses with a status code in the range 500-599 during the last measurement period.	Percent	Since responses with a status code of 500-600 indicate server side processing errors, a high value reflects an error condition.

3.4 The Web Site Layer

The **Web Site** layer is specific to each web site hosted on a web server. An internal WebSite test shown in Figure 2.8 provides a comprehensive view of the states of the individual web sites supported by a web server.

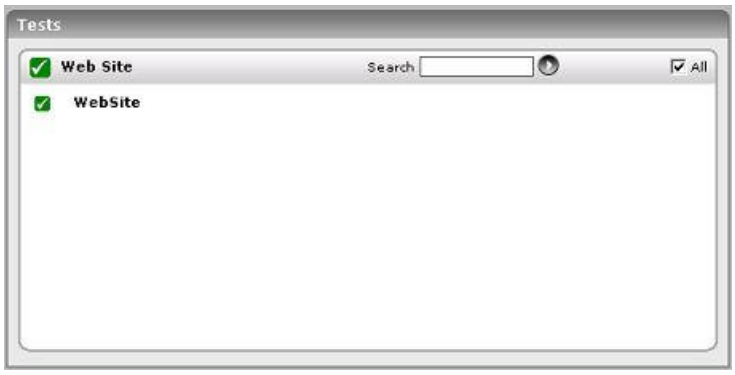


Figure 2.10: The WebSite test tracks the health of the Web Site layer

3.4.1 Web Site Test

For each web site configured for monitoring on the target web server, this test reports the load on the web site and how well the site processes the load.

Purpose	For each web site configured for monitoring on the target web server, this test reports the load on the web site and how well the site processes the load.
Target of the test	A web site supported by a web server
Agent deploying the test	An internal agent
Configurable parameters for the test	1. **TEST PERIOD** - How often should the test be executed 2. **HOST -** The host for which the test is to be configured 3. **PORT -** the port to which the specified **HOST** listens

Outputs of the test	One set of results for every web site monitored		
Measurements made by the test	**Measurement**	**Measurement Unit**	**Interpretation**
	Connections: Rate of connections to the web site.	Conns/Sec	An increase or decrease in the connection rate can represent a change in the user workload.
	Requests: Rate of requests to the web site.	Reqs/Sec	With the advent of HTTP/1.1, multiple requests can be transmitted over the same TCP connection. The ratio of requests per connection can provide an idea of the effectiveness of the HTTP 1.1 protocol.
	Data transmitted: Rate at which the data is transmitted by the web site in response to user requests.	KB/Sec	A large increase in the data transmission rate can be indicative of an increase in the popularity of a web site hosted on the server.
	Data received: Rate at which the data is received by the web site.	KB/Sec	An increase in this value is indicative of an increase in user requests to the web site.
	Errors: Percentage of error responses from the web site.	Percent	Percentage of responses with a 400 or 500 status code.
	400 errors: Percentage of responses with a status code in the range 400-500.	Percent	An unexpected increase in the percentage of responses with status codes in the range 400-499 can indicate a sudden problem at the server.
	Current requests: Number of server threads/processes currently in use for serving requests for a web site (this measurement is not available for Apache web servers).	Number	A majority of the server threads/processes being used simultaneously to serve requests for a web site may be indicative of a server bottleneck caused by the web site.

3.5 The Web Transactions Layer

One of the unique capabilities of the eG Enterprise suite is its ability to monitor individual web transactions.

To monitor web site transactions to an IIS web server executing on Windows 2000/2003, you need to do the following:

p. Enable 'logging' on the target IIS web server;

q. Modify the eG agent configuration to support web transaction monitoring

Both these procedures have been detailed in the *eG Installation Guide*.

To monitor web site transactions to the IIS web server executing on Windows 2008, you need to install and configure **Advanced Logging** on the target IIS web server, soon after configuring a **Web Server** role on the server. This procedure has also been detailed in the *eG Installation Guide*.

Figure 2.11: The tests that map to the Web Transactions layer of a web site.

3.5.1 Web Transactions Test

The Web Transactions test tracks the state of the individual transactions of a web site.

Purpose	To measure the state of individual transactions supported by a web site		
Target of the test	A web site supported by a web server		
Agent deploying the test	An internal agent		
Configurable parameters for the test	1. **TEST PERIOD** - How often should the test be executed 2. **HOST -** The host for which the test is to be configured 3. **PORT -** the port to which the specified host listens		
Outputs of the test	One set of results for each transaction supported by a web site		
Measurements made by the	**Measurement**	**Measurement Unit**	**Interpretation**

test	**Requests:** Rate of requests for a specific transaction.	Reqs/Sec	An increase or decrease in request rate for a specific transaction can represent a change in the user workload.
	Errors: Percentage of error-filled responses from the web site for a specific transaction.	Percent	Percentage of responses for the transaction that report a 400 or 500 status code.
	Data transmitted: Rate at which the data is transmitted by the web site in response to user requests for a specific transaction.	KB/Sec	A large increase in the data transmission rate can be indicative of an increase in the popularity of a specific transaction. Alternatively, a sudden increase in data rate may also indicate a change in the characteristics of a transaction.
	Avg response time: The average time taken by the web site to respond to requests for a specific transaction, measured in seconds. Only requests for which successful responses are received are considered while computing the average response time.	Secs	An increase in response time is a major cause for user dissatisfaction with e-business sites. By correlating an increase in response time with the other metrics collected by the agent per transaction, an operator can diagnose the reason(s) for the deterioration in performance.

Chapter 4

Monitoring Apache Web Servers

The eG Enterprise suite provides a generic *Web server* component-type that allows administrators to effectively monitor an Apache web server, IBM HTTP server, or an iPlanet web server. The eG web adapter can be deployed on a *Web* server component, so as to snoop on the real web transactions to the server and retrieve a wide range of real-time statistics pertaining to the critical transactions that are configured using the eG administrative interface.

Such transaction-related metrics might not interest a few administrators. These administrators would prefer to have overall health indicators such as as the availability of the Apache web server, the data processing ability of the server, the data traffic handled by the server, etc., monitored. To cater to the requirements of such administrators, eG Enterprise provides a specialized *Apache Web* server model (see Figure 4.1).

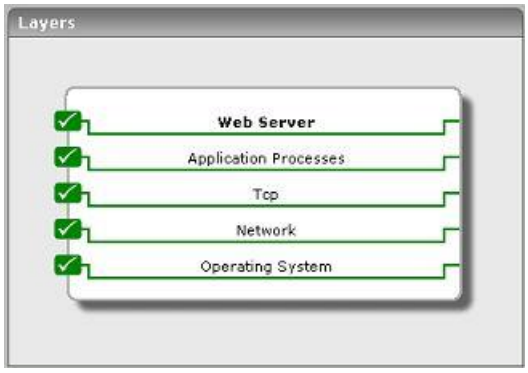


Figure 4.1: Layer model of the Apache web server

Each of the layers depicted by Figure 4.1 is associated with a series of tests that report on key Apache-specific performance parameters, and answers the following performance-related queries:

- Is the web server available?
- How frequently is the web server accessed?
- Is CPU utilization of the web server optimal?
- How is the traffic on the web server?
- How well does the web server process requests? Are there any bottlenecks in processing?

As the **Application Processes**, **Tcp**, **Network**, and **Operating System** layers have been discussed in great detail in the Chapter 1, the sections to come will discuss the **Web Server** layer alone.

4.1 The Web Server Layer

The tests associated with the **Web Server** layer monitor the availability of the Apache web server and measures the traffic on the Apache web server, its current CPU load, how well the Apache server handles requests, etc (see Figure 4.2).

Figure 4.2: The tests associated with the Web Server layer

Since the Http test has been handled previously, the following section will be dealing with the ApacheStatus test only.

4.1.1.1 Apache Status Test

The Apache Status test reveals critical performance statistics pertaining to an Apache web server. This test, upon execution, accesses a specific URL on the Apache server, which contains the required metrics.

Purpose	Reveals critical performance statistics pertaining to an Apache web server		
Target	An Apache web server		
Agent deploying this test	An internal agent		
Configurable parameters for this test	1. **TEST PERIOD** – How often should the test be executed 2. **HOST** - The host for which the test is to be configured. 3. **PORT** - The port to which the specified **HOST** listens 4. **URL** - In the **URL** box, the URL to be accessed by this test for extracting the Apache web server's performance statistics, will be displayed by default. The URL is: *http://{Apache web server host}:{Apache web server port}/server-status*.		
Outputs of the test	One set of results for the Apache web server being monitored		
Measurements	**Measurement**	**Measurement Unit**	**Interpretation**

of the test	**Access rate:** Indicates the rate at which the users access the web server.	Accesses/Sec	
	Data traffic: Indicates the rate at which the web server transmits data.	KB/Sec	
	CPU load: Indicates the percentage of CPU used by the web server.	Percent	If the value of this measure increases with time, it could be a cause for concern.
	Current requests: Indicates the number of requests that are currently being processed by the web server.	Number	
	Idle workers: Indicates number of idle worker threads on the web server, currently.	Number	A very high value of this measure could indicate a processing bottleneck.

Chapter 5

Monitoring Microsoft Transaction Servers

Microsoft Transaction Server (MTS) is widely used in conjunction with IIS in Windows environments. While IIS is used as the front end, MTS is used to host the business logic components. Sporadic or persistent issues in the availability or all-round performance of the MTS server, can delay the processing of business logic, and thus, severely hamper the delivery of the dependent services. This implies that like the IIS web server, the performance of the MTS also deserves constant attention.

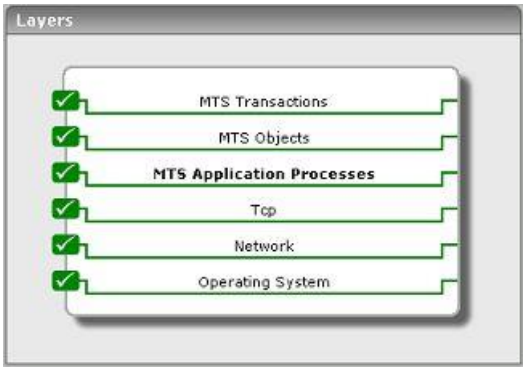


Figure 5.1: Layer model for MTS servers

eG Enterprise provides an exclusive monitoring model for the MTS server (see Figure 5.1), which monitors all integral components of the MTS server, and collects a gamut of performance metrics from the server. These metrics enable administrators to find answers to key performance queries such as the ones discussed below:

- Are the critical MTS processes currently running?
- Are the processes using up too much CPU/memory resources?
- What is the current workload on the MTS server in terms of the number of packages currently running on it?
- How quickly is the MTS server processing transactions?
- Are too many transactions getting aborted too frequently?

- How responsive is the MTS server to requests from clients?

The **Operating System**, **Network** and **Tcp** layers have been already discussed in this chapter. Above the **Tcp** layer is the **MTS Application Processes** layer. The second layer is the **MTS Objects** layer, and the topmost layer is the **MTS Transactions** layer.

5.1 The MTS Application Processes Layer

This layer depicts the states of the different processes that must be executing for the application service to be available. The MtsProcesses test that is similar to the Processes test, tracks various statistics pertaining to the different application processes. The details pertaining to the Processes test have been provided in the *Monitoring Unix and Windows Servers* document.

5.2 The MTS Objects Layer

The **MTS Objects** layer tracks various statistics pertaining to the packets hosted on an MTS server with the aid of Mts test shown in Figure 5.2. The details about this test are given below.

Figure 5.2: Tests mapping to the MTS Objects layer

5.2.1 MTS Test

This test collects statistics pertaining to the packages hosted on a MTS server.

Purpose	This test gives information about the packages hosted on an MTS server.
Target of the test	An MTS server
Agent deploying the test	An internal Agent
Configurable parameters for the test	1. **TEST PERIOD** - How often should the test be executed 2. **HOST -** The host for which the test is to be configured 3. **PORT -** the port to which the specified **HOST** listens
Outputs of the test	One set of results for every MTS server monitored

Measurements made by the test	Measurement	Measurement Unit	Interpretation
	Total packages: Indicates the number of packages installed on the MTS server.	Number	This measure can be used to identify when packages have been installed and uninstalled on the MTS server.
	Total components: Indicates the number of components installed on the MTS server.	Number	This measure can be used to track when components have been installed and uninstalled on the MTS server.
	Running packages: Indicates the number of packages, which are running currently on the MTS server.	Number	This measure gives the current workload on the MTS server.

5.3 The MTS Transactions Layer

This layer employs an **MtsTransactions** test to assess how efficiently the MTS server processes transactions.

Figure 5.3: Test associated with the MTS Transactions layer

5.3.1 MTS Transactions Test

This test monitors the transactions executing on an MTS server.

Purpose	Monitors the transactions executing on an MTS server
Target of the test	An MTS server
Agent deploying the test	An internal Agent
Configurable parameters for the test	1. **TEST PERIOD** - How often should the test be executed 2. **HOST -** The host for which the test is to be configured 3. **PORT -** the port to which the specified **HOST** listens

Outputs of the test	One set of results for every MTS server monitored		
Measurements made by the test	**Measurement**	**Measurement Unit**	**Interpretation**
	Aborted transactions: The number of transactions that were aborted during the last measurement period	Number	A high value indicates a bottleneck on the MTS server.
	Aborted transactions rate: The rate at which the transactions on an MTS server aborted during the last measurement period	Trans/Sec	
	Active transactions: The number of transactions that are currently active	Number	A high value for this measure may indicate a large number of active transactions. Alternately, this may also indicate that due to some reasons the users are not able to complete the transactions.
	Max active transactions: The high watermark of active transactions during the last measurement period	Number	
	Committed transactions: The number of transactions that were committed during the last measurement period	Number	This measure indicates the throughput. A higher value indicates that more transactions are being committed.
	Committed transactions rate: The rate at which transactions were committed during the last measurement period	Trans/Sec	
	Force aborted transactions: The number of transactions that were forcefully aborted during the last measurement period	Number	
	Force committed transactions: The number of transactions that were forcefully committed during the last measurement period	Number	
	In-doubt transactions: The number of transactions in doubt	Number	
	Avg response time: The average time taken by the MTS server to respond to user requests	Secs	A high response time denotes a problem. Poor response times may be due to the server being overloaded or misconfigured.

	Max response time: The high water mark of response times based on replies returned by the server	Secs	If this value is consistently different from the average response time, further investigation of other server metrics may be necessary.
	Min response time: The low water mark of response times based on replies returned by the server	Secs	
	Total transaction rate: The number of transactions performed per second	Trans/Sec	A high value of this metric indicates a lot of transactional activity happening on the server.

Chapter 6

Monitoring Oracle 9iAS Http Servers

Oracle HTTP Server (OHS) is a web server that provides the necessary infrastructure to support dynamic applications. Based on the Apache infrastructure, Oracle HTTP Server allows developers to program their site in a variety of languages and technologies.

In environments where the OHS is used to support critical end-user services, it is imperative to ensure that the OHS is continuously functioning to peak capacity, and is available 24x7 to serve client requests. Intermittent web server failures and even slight dips in the responsiveness of the server, could significantly delay, and sometimes even halt, service delivery, causing the business to lose millions. To avoid such adversities, the internal health and availability of the OHS needs to be constantly monitored.

eG Enterprise provides a specialized monitoring model for the Oracle HTTP server (see Figure 6.1), which runs periodic health checks on the server, determines its current state, detects performance issues (if any), and proactively alerts administrators to these issues, so that they can hasten the remedial measures.

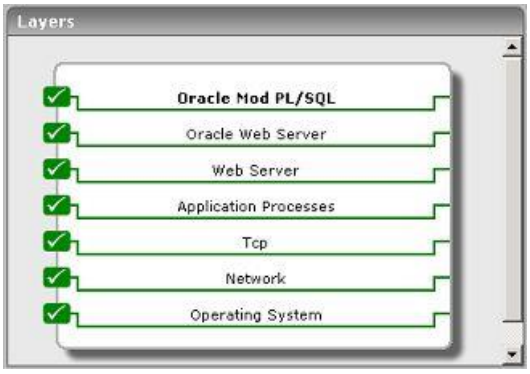


Figure 6.1: The layer model of an Oracle HTTP server

This model constitutes a set of hierarchical layers, which are mapped to a wide variety of tests. When the eG agent on the Oracle HTTP server executes these tests, it pulls out a wealth of performance information from the server. In the light of the statistics so retrieved, administrators can find answers to the following performance queries:

- Is the web server responding quickly to user requests?
- What is the current load on the server in terms of data traffic?

- Do errors occur on the web server? If so, how frequently do they occur, and what are the corresponding error codes?
- Are the PL/SQL Gateway's session and content caches utilized effectively, or are there too many cache misses?
- How well does the PL/SQL gateway respond to user requests? Are there too many errored responses?

Note:

Any component on an Oracle 9i Application server (be it OC4J or the Oracle HTTP server) can be monitored by eG only if the Oracle 9i Application Server Release 2.0 is installed, with dmctl and dmstool.

The sections to come deal only with the top 2 layers of Figure 6.1, as all other layers have been dealt with extensively in the previous chapters.

6.1 The Oracle Web Server Layer

Besides measuring the responsiveness of the web server, this layer also reveals the internal health of the server by reporting on its request processing ability, the data traffic to the server, etc.

Figure 6.2: The tests associated with the Oracle Web Server layer

6.1.1 Oracle HTTP Server Test

This test reports the performance statistics pertaining to an Oracle Http server.

Purpose	Reports the performance statistics pertaining to the Oracle Http server
Target of the test	An Oracle HTTP server
Agent deploying the test	An internal Agent

Configurable parameters for the test	1. **TEST PERIOD** - How often should the test be executed 2. **HOST -** The host for which the test is to be configured 3. **PORT -** The port to which the specified **HOST** listens 4. **HOMEDIR** – The path to the directory in which the Oracle 9i application server has been installed		
Outputs of the test	One set of results for every Oracle HTTP server monitored		
Measurements made by the test	**Measurement**	**Measurement Unit**	**Interpretation**
	Response time: Indicates the time taken by the web server to respond to requests. Only requests for which successful responses are received are considered while computing the response time.	Secs	A very high value for this measure indicates a bottleneck on the web server or one of the applications on the server.
	Requests: Indicates the rate of requests to the web server.	Reqs/Sec	This value is indicative of the server workload.
	Active connections: Indicates the number of connections currently open to the server.	Number	A high value can indicate a slow-down of the web server or applications hosted by it.
	Connection rate: Indicates the rate of connections to the web server during the last measurement period.	Conns/Sec	An increase or decrease in connection rate can represent a change in the user workload.
	Data transmit rate: Indicates the rate at which the data was transmitted by the server to clients during the last measurement period.	KB/Sec	A large increase in the data transmission rate can be indicative of an increase in the popularity of one or more web sites hosted on the server.
	Errors: Indicates the rate at which errors occurred on the server during the last measurement period.	Errors/Sec	The details of the type of errors occurring are reported by the test OracleHttpResponse test.
	POST requests: Indicates the rate of post requests to the web server.	Reqs/Sec	
	GET requests: Indicates the rate of get requests to the web server.	Reqs/Sec	

6.1.2 Oracle HTTP Response Test

This test measures the request-handling capability of the Oracle9iAS HTTP server.

Purpose	Measures the request-handling capability of the Oracle HTTP server		
Target of the test	An Oracle HTTP server		
Agent deploying the test	An internal Agent		
Configurable parameters for the test	1. **TEST PERIOD** - How often should the test be executed 2. **HOST -** The host for which the test is to be configured 3. **PORT -** The port to which the specified **HOST** listens 4. **HOMEDIR** – The path to the directory in which the Oracle 9i application server has been installed		
Outputs of the test	One set of results for every Oracle HTTP server monitored		
Measurements made by the test	**Measurement**	**Measurement Unit**	**Interpretation**
	200 responses: Indicates the percentage of requests that were successfully serviced by the web server during the last measurement period.	Percent	
	Errors: Indicates the percentage of responses with errors during the last measurement period.	Percent	Responses with a 400 or 500 status code are counted as errors.
	300 responses: Indicates the percentage of responses in the last measurement period with a status code in the range of 300-399.	Percent	300 responses could indicate page caching on the client browsers or redirection of requests. A sudden change in this value is indicative of a problem condition.
	400 errors: Indicates the percentage of responses in the last measurement period with a status code in the range of 400-499.	Percent	A high value indicates a number of missing/error pages.
	500 errors: Indicates the percentage of responses in the last measurement period with a status code in the range of 500-600.	Percent	Since responses with a status code of 500-600 indicate server side processing errors, a high value reflects an error condition.

6.2 The Oracle Mod PL/SQL Layer

The PL/SQL Gateway provides support for building and deploying PL/SQL-based applications on the web. PL/SQL stored procedures can retrieve data from database tables and generate HTTP responses containing data and code to display in a Web browser.

This layer assesses how effectively the gateway uses the content and session caches while processing requests, and keeps track of the responses the gateway sends out to a web browser.

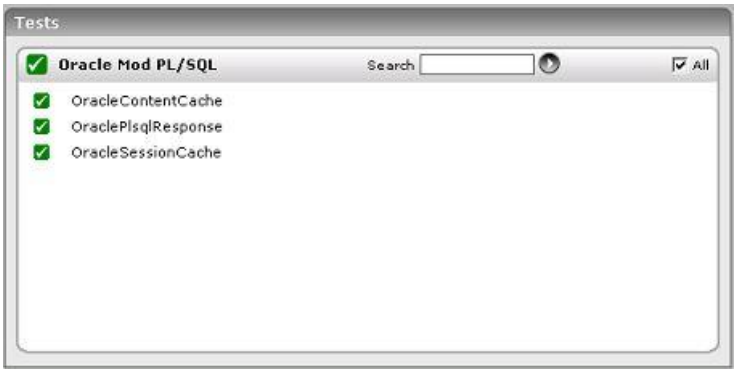


Figure 6.3: Tests associated with the Oracle Mod PL/SQL layer

6.2.1 Oracle Content Cache Test

To improve performance of PL/SQL web applications, you can leverage the caching feature provided by the PL/SQL Gateway. This feature allows caching of PL/SQL procedure Web content in the middle-tier. Subsequent requests for the content may be retrieved from the cache, with or without validation from the database, thereby decreasing the database workload.

This test reports the performance metrics pertaining to the mod_plsql gateway's content cache.

Purpose	Reports the performance metrics pertaining to the mod_plsql gateway's content cache.		
Target of the test	An Oracle HTTP server		
Agent deploying the test	An internal Agent		
Configurable parameters for the test	1. **TEST PERIOD** - How often should the test be executed 2. **HOST -** The host for which the test is to be configured 3. **PORT -** The port to which the specified **HOST** listens 4. **HOMEDIR** – The path to the directory in which the Oracle 9i application server has been installed		
Outputs of the test	One set of results for every Oracle HTTP server monitored		
Measurements made by the	**Measurement**	**Measurement Unit**	**Interpretation**

test	**Request rate:** Indicates the rate of requests to the content cache during the last measurement period.	Reqs/Sec	A very high value for this measure indicates that the content cache is handling many requests.
	Cache hits: Indicates the rate of content cache hits.	Conns/Sec	Reading from the cache is less expensive than reading directly from the database. Therefore, the higher this value, the better. Remember that caching increases the scalability of a web application.
	Cache misses: Indicates the rate of content cache misses	Conns/Sec	

6.2.2 Oracle Session Cache Test

This test reports the performance metrics pertaining to the mod_plsql gateway's session cache.

Purpose	Reports the performance metrics pertaining to the mod_plsql gateway's session cache		
Target of the test	An Oracle HTTP server		
Agent deploying the test	An internal agent		
Configurable parameters for the test	1. **TEST PERIOD** - How often should the test be executed 2. **HOST -** The host for which the test is to be configured 3. **PORT -** The port to which the specified **HOST** listens 4. **HOMEDIR** – The path to the directory in which the Oracle 9i application server has been installed		
Outputs of the test	One set of results for every Oracle HTTP server monitored		
Measurements made by the test	**Measurement**	**Measurement Unit**	**Interpretation**
	Request rate: Indicates the rate of requests to the session cache during the last measurement period.	Reqs/Sec	A very high value for this measure indicates that the session cache is handling many requests.
	Cache hits: Indicates the rate of content cache hits.	Conns/Sec	Reading from the cache is less expensive than reading from the database. Therefore, the higher this value, the better. Remember that caching increases the scalability of a web application.

	Cache misses: Indicates the rate of content cache misses.	Conns/Sec	

6.2.3 Oracle PL/SQL Response Test

This test reports how well the mod_plsql gateway of the Oracle9iAS HTTP server responds to user requests.

Purpose	Reports how well the mod_plsql gateway of the Oracle9iAS HTTP server responds to user requests.		
Target of the test	An Oracle HTTP server		
Agent deploying the test	An internal Agent		
Configurable parameters for the test	1. **TEST PERIOD** - How often should the test be executed 2. **HOST -** The host for which the test is to be configured 3. **PORT -** The port to which the specified **HOST** listens 4. **HOMEDIR** – The path to the directory in which the Oracle 9i application server has been installed		
Outputs of the test	One set of results for every Oracle HTTP server monitored		
Measurements made by the test	**Measurement**	**Measurement Unit**	**Interpretation**
	200 responses: Indicates the percentage of requests that were successfully serviced by the web server during the last measurement period.	Percent	
	Errors: Indicates the percentage of responses with errors during the last measurement period.	Percent	Responses with a 400 or 500 status code are counted as errors.
	300 responses: Indicates the percentage of responses in the last measurement period with a status code in the range of 300-399.	Percent	300 responses could indicate page caching on the client browsers or redirection of requests. A sudden change in this value is indicative of a problem condition.
	400 errors: Indicates the percentage of responses in the last measurement period with a status code in the range of 400-499.	Percent	A high value indicates a number of missing/error pages.

	500 errors: Indicates the percentage of responses in the last measurement period with a status code in the range of 500-600.	Percent	Since responses with a status code of 500-600 indicate server side processing errors, a high value reflects an error condition.

Chapter 7

Externally Monitoring Web Servers

There is no doubt that it is imperative to monitor the request processing ability of the web server, the amount of data load that a web server can handle, or the key transactions to a web site hosted by the server. In fact, eG Enterprise addresses these critical internal monitoring needs using dedicated web server monitoring models (*IIS*, *Apache Web*, etc.), and its unique web adapter technology, all of which have been discussed previously. However, sometimes, administrators might only be interested in knowing whether the web server is available or not, and if so, how responsive it is to user requests. Some other administrators may not have access to the web servers to install agents, but may want to check its availability and responsiveness. To cater to such monitoring needs, eG Enterprise offers the exclusive, *External Web* model (see Figure 7.1). This model requires only an external agent, which employs native application-level protocols, to ascertain the availability and responsiveness of the web server.

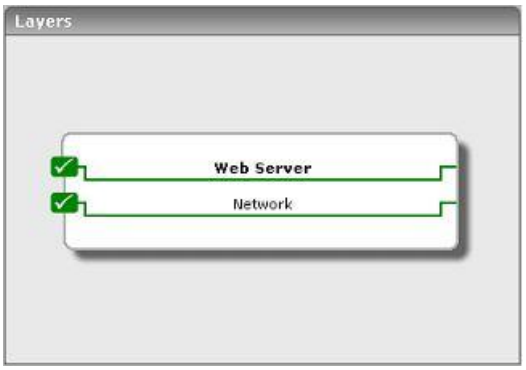


Figure 7.1: Layer model of the External web server

The **Network** test executed by the **Network** layer of Figure 7.1 is a typical external test that reports whether/not a healthy network connection exists between the agent host and the target web server. The **Http** test that executes on the **Web Server** layer emulates a user accessing a web server, and determines the availability, responsiveness, and the errors occurring on the web server, from an external perspective.

Please refer to the previous chapters for more details on the Network test and Http test.

Chapter 8

Conclusion

This document has described in detail the monitoring paradigm used and the measurement capabilities of the eG Enterprise suite of products with respect to **web servers**. For details of how to administer and use the eG Enterprise suite of products, refer to the user manuals.

We will be adding new measurement capabilities into the future versions of the eG Enterprise suite. If you can identify new capabilities that you would like us to incorporate in the eG Enterprise suite of products, please contact support@eginnovations.com. We look forward to your support and cooperation. Any feedback regarding this manual or any other aspects of the eG Enterprise suite can be forwarded to feedback@eginnovations.com.