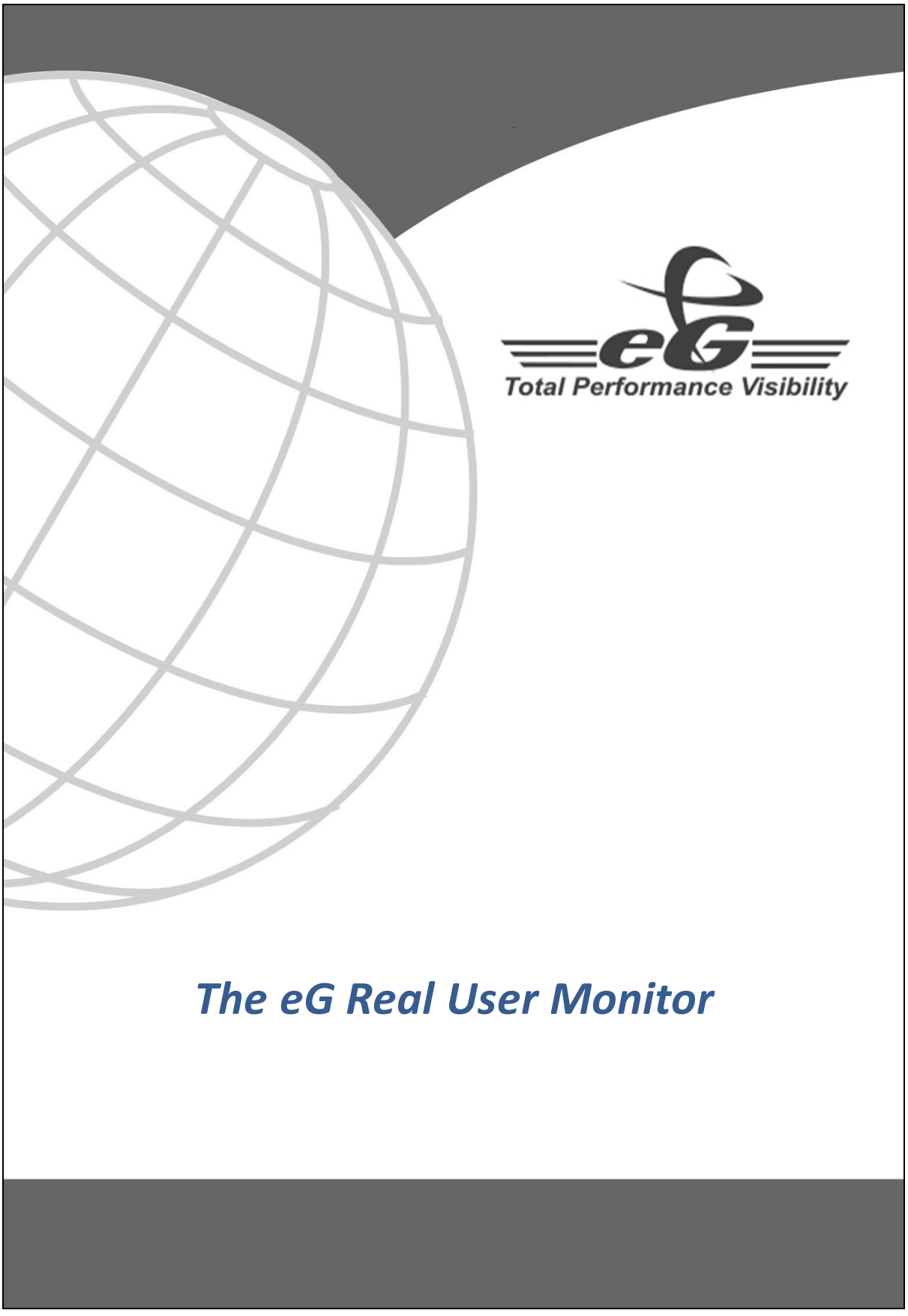


The eG Real User Monitor

Table of contents

Table of Figures

1

Introduction

During the last decade, many businesses morphed into e-Businesses that offered their services to end-users over the World Wide Web. On top of advanced web infrastructures, enterprises hosted web sites and/or web applications that took their service offerings straight into the homes of customers! Vide these web-based windows, end-users could access any service – be it retail banking, shopping, or ticket booking – from the comfort of their couches, using just a browser and a mouse! Businesses too thrived, as the web significantly widened the reach of the services and increased revenues. The rise in popularity of web-based applications also bought about a paradigm shift in how application performance is monitored and managed.

Most web site/web applications overlay multiple inter-related IT infrastructure tiers – the web server front end, the business logic middle tier, the network tier, the backend storage tier, etc. For long, web site/ web application performance was measured only in terms of the availability and resource usage of the individual infrastructure tiers at work – is the web server available? is the application server sized right? does the database server have enough disk space? In recent times however, the emphasis on 'service quality' and the huge penalties attached to 'service level' violations, caused the performance management focus to shift from these "servers" to the "users". But, why "users"? This is because, unlike IT administrators who are solely concerned with 'server' performance, users are the ones who provide a 'service perspective to performance'. Typically, when problems occur in an IT infrastructure, an administrator will only want to know which 'servers' are affected and what kind of problems are affecting 'server' performance. These administrators are rarely ever concerned with how these 'server' problems impact the performance of the web site/web applications they support! End-users on the other hand, view any web site/web application they work with as a 'service' offered to them by an enterprise. When end-users run into issues while working with a web site/web application, they instantly tag them and flag them as 'service' issues. For instance, if say, a banking web site slows down, the corresponding user complaint will read as follows: the *banking service is slow*. In many cases, user may not even know that multiple 'servers' are in fact working together in the background to deliver the banking service. If IT is able to see what a user sees, it will be able to proactively identify a potential service outage, determine its root-cause, take rapid steps to avert it, and thus save millions by way of penalties! High service uptime also results in many happy users, which naturally translates into a wider customer base, increased revenues, and continued goodwill for the business. On the other hand, if poor user experience with a service goes undetected, user dissatisfaction with a service will persist, causing penalties to rise, reputation to be damaged, and revenues to dwindle. In short, the success/failure of a business today greatly depends on the user experience with its web services. It is hence imperative that IT administrators continuously monitor and efficiently manage user experience with their critical web services and detect the unavailability of the service and probable slowdowns in service delivery well before users do.

1.1 How to Measure End User Experience?

Traditionally, end user experience was measured using synthetic user transactions. Every step of a user interaction with a web site/web application was recorded and periodically played back from different locations to check the availability and responsiveness of each step. The exact step at which user experience took a

beating can be isolated in the process. What's questionable about this approach though is its reliability! Since the user experience metrics captured using this approach are based on 'simula ted' transactions and not 'actual' user transactions, you may end up having user experience measured even when there is no real load on the service!

This is exactly where **Real User Monitoring** scores! This is a passive monitoring technology that records all **real** (not emulated) **user** transactions to a website or web application. In the process, response time metrics are collected for every transaction of each user in real-time. When there is no load, response time will not be captured! This way, the **Real User Monitor** captures the true user experience with a web site/web application.

Though an ideal user experience monitor would be one that supports both synthetic and real user monitoring, for accurate insights into actual transaction load and to know what a user really sees, **Real User Monitoring** is essential.

1.2 How does eG Enterprise Monitor End-User Experience with Web Sites/Web Applications?

eG Enterprise is one of those rare monitoring solutions that provides *multi-dimensional end-user experience monitoring*. Using eG, user experience with web-based services can be measured using both *synthetic* and *real user* transactions.

eG Enterprise joins hands with third-party emulation tools such as Tevron's Citra Test and Itexis AppsMon, to record every step of a user interaction with a web site/web application, and plays back the recording at configured intervals to capture the availability and response time of each step.

Complementing the *transaction emulation approach,* is eG's *real user monitoring* approach! eG's Real User Monitor tracks LIVE, the transactions of real users (not emulated) to web sites/web applications, and measures the response time of each transaction – i.e., page view - in real-time. In the process, the eG Real User Monitor promptly alerts administrators to slow page views, and isolates the exact tier where user experience was bottlenecked – was it at the front end tier? backend tier? Or the network tier?

In this document, we will be elaborating on eG's Real User Monitoring capability only.

2

Real User Monitoring Using eG Enterprise

To be able to understand and appreciate eG's approach to real user monitoring, it is imperative that you first understand the typical qualities a RUM (Real User Monitor) tool should possess and what's lacking in traditional RUM tools. The sections that follow will shed light on these subjects.

2.1 Qualities of an Ideal Real User Monitor

There are certain qualities that one should look for in a Real User Monitor:

- **Easy to install/maintain:** The Monitor should not require expert assistance for deployment. Moreover, it should install quickly and should not involve cumbersome maintenance.
- **Non-intrusive monitoring:** The Monitor should capture responsiveness of real user transactions in a non-intrusive manner - in other words, it should not mandate the installation of agent software on business-critical servers for monitoring. It should also not interfere with server operations or adversely impact server performance in any way.
- **Capture responsiveness regardless of development framework and programming language of web applications:** Java, .NET, HTML are some of the widely used languages for building web applications. Each of these languages in turn runs on robust development frameworks. A good Real User Monitor is one that can snoop on transactions to any web application, regardless of the programming language and framework on which it has been developed.
- **Accurately detect the root-cause of poor user experience:** An ideal Real User Monitor is one that not just pinpoints transactions that are slow, but reveals where the transactions could have been bottlenecked – at the front-end? the backend? Or the network? This hastens problem resolution and ensures high service availability.
- **Measure user-experience from different browser/browser versions:** Sometimes, what browser you use to access a web application may impact responsiveness. The Real User Monitor should be able to report user experience as seen from different browsers/browser versions, so that IT is able to identify the browsers on which their web applications work best.
- **Reveal how response time varies with geography:** Where the infrastructure spans multiple geographies, it is only natural for administrators to expect the Real User Monitor to capture user experience from every geography, so that they can quickly mark the 'problem zones' in the infrastructure. The service interactions of users in such geographies can be closely scrutinized to figure out the reason for the high response time.
- **Identify popular clients:** Web applications these days are accessed from a variety of client devices – desktops, mobile phones, tablets, are just to name a few. Enterprises today want to know the device that is

used by a large majority of their web application users, so that they can tune the look, feel, and performance of their web applications to satisfy such users and ensure their loyalty. At the same time, they also want to quickly compare responsiveness across the device types, so that they know which devices are seeing poor responsiveness and how many users are affected in the process. The Real User Monitor should provide such device-specific insights.

- **Capture Javascript errors**: Javascript errors are also key spoilers of the user experience with a web application. Any RUM solution should therefore be able to capture and report such errors, as soon as they occur.

2.2 Challenges Faced when Using Traditional RUM Tools

Some of the most common tools used for real user monitoring are as follows:

- **Network Probes/Sniffers:** These are hardware devices that can be deployed on the network, connected to span ports, and configured to snoop on traffic over the network to capture and analyze user experience metrics in real-time.
- **Client Agent:** This approach involves installing an agent on the client device and configuring that agent to track the requests and responses at the client.

Though network probes are capable of monitoring non-intrusively without adding to the load on the network or the client, deployment of the probes is a challenge as it will require the approval and assistance of networking experts. Moreover, the probes may not be able to obtain the full application context, as network transmissions are often encrypted. Using a client agent too is not without disadvantages. Installing and later maintaining the agent software on every client can prove to be a daunting task, particularly in environments characterized by hundreds of users connecting from numerous clients.

Owing to these limitations, neither of these tools qualify as ideal Real User Monitors.

2.3 The eG Approach to Real User Monitoring

eG Enterprise employs a proven, time-tested, 'Java script' approach to monitoring real user transactions to web sites/web applications. This is an 'agentless approach' that involves embedding a small Java script in every page to be monitored in the target web site/web application. Whenever the browser loads one of these web pages in response to a user request, the script runs and collects the following metrics:

- The page load time;
- Break-up of page load time;
- Count of Java script errors (if any);
- Which browser was used to access the page? What is the browser version?
- On what operating system is the browser running?
- From which device the request came in – desktop? mobile phone? or tablet?

Upon metrics collection, the browser sends performance beacons carrying the gathered metrics to a software component called the eG RUM Collector.

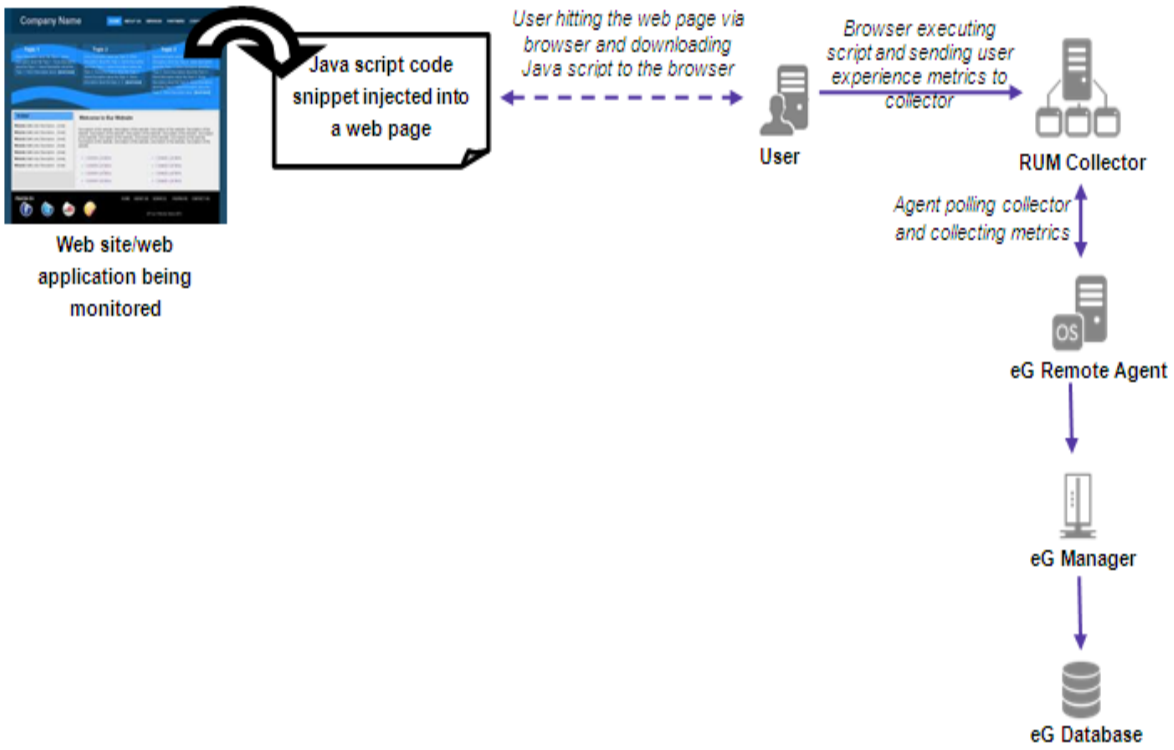


Figure 2.1: How the eG RUM works

The collector receives the metrics and stores them locally in flat files. Additionally, the collector also intelligently determines the geography (country, city, and region) from where the request originated and stores this data too in the flat files. An eG agent deployed on a remote host, can then be configured to poll the RUM collector at regular intervals to collect the request and response time metrics and the details of the geography. The remote eG agent then aggregates the collected metrics based on:

- The web site/web application monitored
- URL groups configured for monitoring
- Transaction URL patterns configured for monitoring
- Geographies – i.e., countries, cities, and regions
- Page types – Page, Iframe, AJAX
- Browsers
- Devices

The aggregated results are then compared with thresholds to isolate deviations. Both the aggregated metrics and the threshold violations are then reported to the eG manager, which presents the performance and problem information in the eG monitoring console and also stores the same in the eG backend.

2.4 What the eG RUM Reveals

Using the metrics reported by eG RUM, administrators can find quick and accurate answers to the following performance queries:

- How is the traffic to your web site/web application? Is it abnormally high or low?
- Are the pages in the web site/web application loading quickly? If not, then what is causing the slowness - is it the network? front end? or backend?
- If the frontend is causing page loading to slowdown, then what is the source of the slowness with the front end? - is it because of a delay in page rendering, DOM downloading, or in DOM processing?
- If the problem is with the network, then what is the exact network issue? - is there a delay in domain name resolution? Or is there a poor network connection to the server?
- How many page views have satisfied users in terms of page load time, and how many have not? Which are the slow pages, and where does the bottleneck lie?
- Have any pages encountered JavaScript errors? If so, which ones are they and what are those errors?
- What is the Apdex score of the web site/web application? Is it satisfactory or not?
- Which specific page groups in the web site/web application are seeing the maximum user traffic?
- Are users to a particular page group experiencing more slowness than the others? Which page group is it? Which pages in the page group are contributing to this slowness?
- Are iFrames loaded quickly?
- Is AJAX code processed rapidly?
- Are the base pages loading swiftly?
- Which are the devices that are being used by users to the web site/web application? How many page requests are coming from each device?
- Are users using a specific device experiencing more performance issues than the rest? If so, which device is it?
- What are the browsers that are being used by users to the web site/web application? How does the experience of users vary with browser? Is any browser seeing more performance issues than the rest? If so, which browser is it?

- In which countries, cities, and regions is your web site/web application most popular?
- Are users connecting from a particular country, city, or region experiencing more slowness than the rest?

2.5 The eG RUM Advantage

What gives eG RUM an edge over its competition are its unique features:

- **Hassle-free installation:** It takes only a few minutes to install and configure the RUM collector, which is the core component of the eG RUM architecture. Moreover, eG auto-generates the Java script to be injected into the monitored web pages, thus saving coding time and the need for expert programmers. Once the collector is started and the code injected, no additional instrumentation is necessary to make the browser send performance beacons to the collector.
- **Hands-free monitoring:** eG RUM is an 'agentless' solution. The eG RUM collector and the eG Agent can both be deployed on resource-thin, less-critical hosts in the environment. This way, you can make sure that your mission-critical servers/applications are protected from any performance-impacting third-party intrusions. Moreover, minimal firewall configurations are required to allow the monitoring. For instance, to enable the agent to remotely poll the eG RUM collector for metrics, the firewall needs to allow only one-way communication between the agent and the collector. This way, you can get the solution up and running in no time and with minimal manual intervention.
- **Real-time monitoring of real user transactions:** You can configure specific transaction URLs for monitoring by eG RUM. If this is done, then, when a user actually requests for such transactions, the eG RUM snoops on the LIVE transactions and captures their responsiveness. Slow transactions can thus be accurately isolated. Moreover, since no emulation techniques are used for transaction monitoring, you are sure to receive the true picture of transaction performance.
- **Language/Framework-independent monitoring:** The eG RUM is capable of monitoring the responsiveness of any web site/web application – regardless of the language/development framework beneath. This implies that the eG RUM can track requests to Java, HTML, JavaScript, and even .NET pages that are part of a web site/web application, and report the responsiveness of these pages.
- **Precise root-cause identification of poor user experience:** Many factors affect user experience with a web service – a congested network connection, backend application and database servers with low processing power, and an unavailable/overloaded front-end, are some of the key factors. When a user complaints that his/her experience with a web service is below par, administrators must be able to quickly identify the exact transaction that is slow and what could be causing the slowness. This is where the eG RUM helps! Using the eG RUM, administrators can not only identify slow transactions, but can accurately figure out where the transaction is bottlenecked – at the network? the front-end? or the backend? Precise problem identification enables quick problem resolution and consequently, high service uptime.
- **Monitors browsers and their impact on transaction performance:** The eG RUM provides browser-specific insights into responsiveness of web transactions, and in the process, points you to those browsers that are often seeing poor responsiveness. When building a web application, these insights will help you in identifying the browsers that are incompatible with your web applications, and those that are ideal for use with your web applications.

- **Auto-discovers where (i.e., the location) application users are coming from and reports experience per location:** Using the eG RUM, you can quickly know from which geography - i.e., country, city, and region – the web application receives requests. Response time metrics are also reported per geography, thus leading administrators to those geographies that are problematic and the reason for the problems. Zooming into a "problem zone" eases troubleshooting, as it helps firmly fix "responsibility centers" and focus troubleshooting efforts on infrastructure elements within a particular geography. This in turn hastens root-cause diagnosis and ultimately, problem resolution.

- **Helps improve the experience of client devices with applications:** Using the eG RUM, administrators can quickly understand the client device being used by a majority of their application users. Administrators can thus instantly detect accesses from unsupported/incompatible devices. Besides, the response time metrics per client device currently used are also reported by the eG RUM, with the help of which, application developers can swiftly and accurately mark troublesome devices. These device-specific performance insights also provide application developers with effective pointers to tweak the monitored application, so as to improve the usability of the application from the problematic devices and thus minimize or completely mitigate complaints from these device users.

- **Captures Java script errors:** The eG RUM instantly alerts administrators to Java script errors that may be encountered when working with a web application/web site. The complete details of these errors are also provided to enable quick and efficient troubleshooting.

It is thus evident that eG possesses all qualities of an ideal Real User Monitor!

3

How to Use the eG Real User Monitor

In order to use the eG Real User Monitor, the following broad steps should be applied:

1. Make sure the pre-requisites for Real User Monitoring are in place.
2. Install, configure, and start the eG RUM collector
3. Manage the web site/web application to be monitored
4. Insert the code snippet provided by eG in every web page to be monitored
5. Modify the test configuration (if required)
6. View the user experience metrics.

3.1 Pre-requisites for Real User Monitoring using eG

Before attempting to monitor real user transactions to your web site/web application, ensure that the following are in place:

- At least one HTTP/HTTPS web site/web application receiving user traffic
- Users connecting to the target web site/web application using any **Javascript-enabled** browser (on desktops/mobile phones/tablets) that supports the **Navigation Timing API**. The following browsers are known to support this API:
- Internet Explorer v9 (and above)

  Note:

  Built-in security policies of Windows Server Editions (eg., Windows 2008, Windows 2012, etc.) will not allow the Internet Explorer browser deployed on them to send beacons to the RUM collector. This means that the eG Real User Monitor will not be able to measure the experience of those users who are using IE on any Windows Server Edition to browse the target web site/web application.
- Edge
- Opera v30 (and above)
- Chrome v31 (and above)
- iOS Safari v9
- Firefox v38 (and above)
- Android Browser v4.1 (and above)
- Chrome for Android v42
- Safari v8 (and above)

- At least one eG RUM collector
- At least one remote agent operating on any remote Windows/Linux/Solaris host in the environment;

Minimum hardware requirements of the eG remote agent:

- Minimum 4GHz of CPU
- Minimum 4GB RAM
- Minimum 1 GB free disk space

  Note:

  - The hardware requirements described above are the minimum requirements for performing real user monitoring using eG. The actual resource requirements will vary depending upon the traffic to your web site/web application. To understand the actual resource requirements for RUM, use the *RUM Sizing Calculator* .
  - The *RUM Sizing Calculator*, as the name suggests, is specific to eG RUM. In other words, if you intend using a dedicated eG remote agent for performing real user monitoring in your environment, then it makes sense to use just the *RUM Sizing Calculator* to determine the amount of resources that you will have to provision on the target agent host. However, if you plan to configure a single eG remote agent to monitor multiple components including the eG *Real User Monitor*, then, you will have to use both the *RUM Sizing Calculator* and the *eG Manager & Database Sizing Calculator* to determine the total resource requirement of the agent host.
- An eG database that is sized right to handle the load generated by the eG RUM; a minimum of 2GB of disk space should be free on the eG database.

  Note:

  - The disk space requirements provided above are the minimum requirements for performing real user monitoring using eG. The actual resource requirements will vary depending upon the traffic to your web site/web application. To understand the actual database space requirement for your environment, use the RUM Sizing Calculator .
  - The database space requirement cited above is RUM-specific. In other words, if you are only performing real user monitoring in your environment, then it would suffice if your eG database is sized with a minimum of 2 GB of free disk space. However, if other components are also being monitored in your environment, then, you will have to use the eG Manager and Database Sizing Calculator and the RUM Sizing Calculator to compute the total resource requirement for your environment.

3.2 Installing and Configuring an eG RUM Collector

A default RUM collector is bundled with the eG manager. By mapping this default collector with any web site/web application (in eG terminology, this refers to the **Real User Monitor** component) to be monitored, you can ensure that performance beacons carrying the user experience metrics related to that web site/web application are sent to the default collector.

Note:

Since the default RUM collector is bundled with the eG manager, stopping the eG manager will stop the collector as well.

Where many web sites/web applications are managed for user experience monitoring, a single collector may not be able to handle the measure load generated. To avoid overloading the default collector, you may want to install and configure multiple collectors and distribute the monitoring load uniformly across all collectors – say, by assigning a separate web site/web application to each collector.

This section describes how to install and configure additional RUM collectors.

3.2.1 Pre-requisites for Installing an eG RUM Collector

- A physical server/VM running Tomcat v6 (or above) on any of the following operating systems:
- Windows 2008/Vista/7/8 (64-bit)
- Windows 2012 (64-bit)
- Solaris 7(or higher)
- Red Hat Linux v3 (or above)
- openSUSE 11 (or above)
- CentOS v5.2 (or above)
- Fedora Linux
- Oracle Linux v6.x (or higher)
- AIX 4.3.3 (or higher)
- HP-UX 10 (or higher)
- FreeBSD 5.4
- Tru64 5.1
- JDK 1.6 or above

The minimum resource requirements of the collector host are as follows:

- 4GHz CPU
- Tomcat heap memory of 512 MB
- 4 GB RAM
- 1 GB disk space

Note:

The hardware requirements described above are the minimum resource requirements for the eG RUM collector. The actual resource requirements will vary depending upon the traffic to your web site/web application. To understand the actual resource requirements for your environment, use the *RUM Sizing Calculator.*

3.2.2 Where to Install the eG RUM Collector?

Where the RUM collector should be deployed and how it communicates with the browsers and the eG remote agent depends upon the following factors:

- Where the users to the target web site/web application are coming from – i.e., where the browsers are;
- Where the eG remote agent is;

Note:

To ensure that the eG manager host is not overloaded, it is recommended that you do not install the collector on the same host as the eG manager.

If a web site/web application being monitored is being accessed by both internet and intranet users (see Figure 3.1), then the eG RUM collector can be hosted on the cloud. In this case, the eG RUM collector should be configured with both a private and public IP address. While the browsers used by internet users will send the performance beacons to the collector via an HTTP/S connection to the collector's public URL, the browsers used by intranet users will send the same using the collector's private URL. Moreover, if the remote agent has been deployed on a host in the intranet, then the agent too will poll only the private URL of the collector (via HTTP/S) to collect metrics.

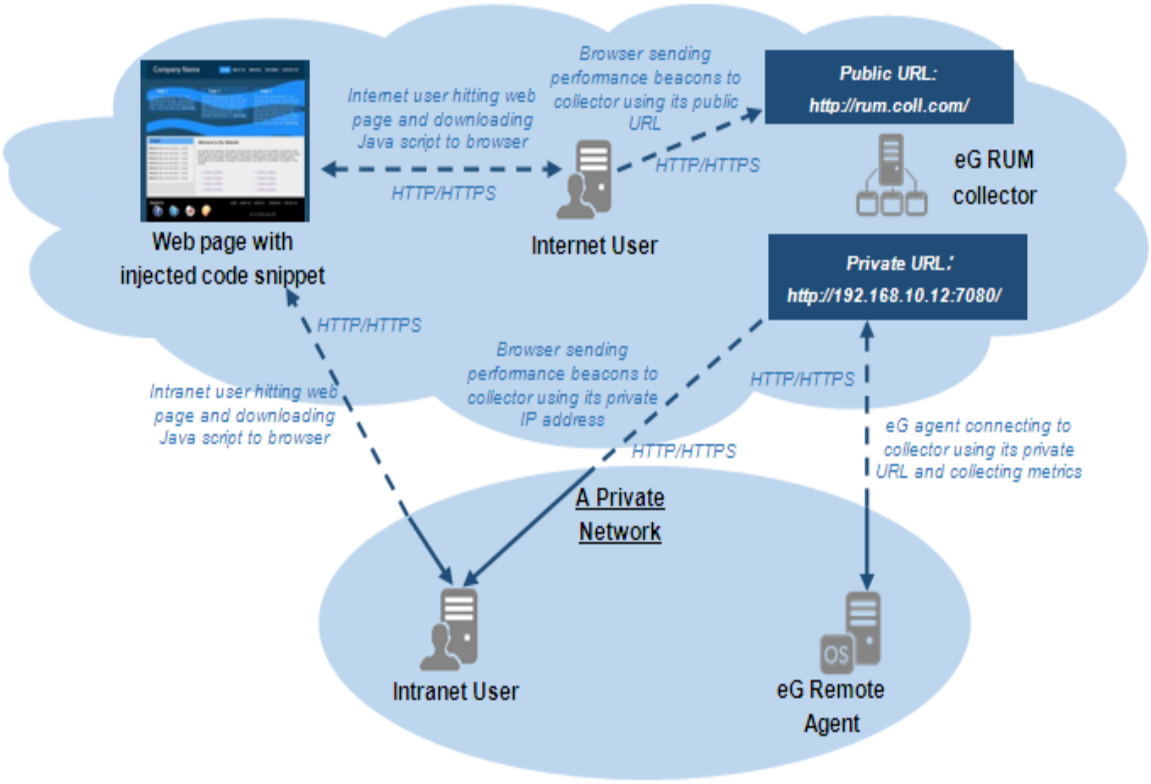


Figure 3.1: A RUM collector deployment model where the target web application is accessed by both internet and intranet users and the eG agent is in a private network

If a web site/web application being monitored is being accessed by internet users only, then the collector should only be hosted on the cloud (see Figure 3.2). In this case, if the eG agent is also deployed on the cloud, it would suffice to configure the collector with a public IP address alone, as both the beacons and the agent will communicate only with the public URL of the collector, via HTTP/S.

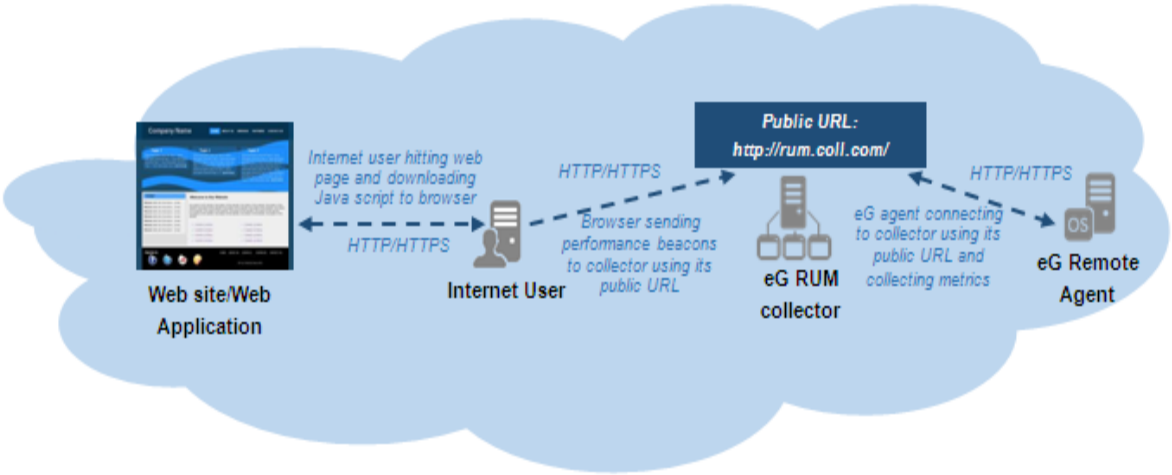


Figure 3.2: A RUM collector deployment model where both the target web application is used only by internet users and the eG agent is on the cloud

However, if in the above scenario, the eG agent alone is in a private network, then the collector should be configured with both a public and private IP address (see Figure 3.3). In this case therefore, the browsers will communicate metrics to the collector via an HTTP/S connection to its public URL, and the eG agent will communicate with the collector's private URL to gather the metrics.

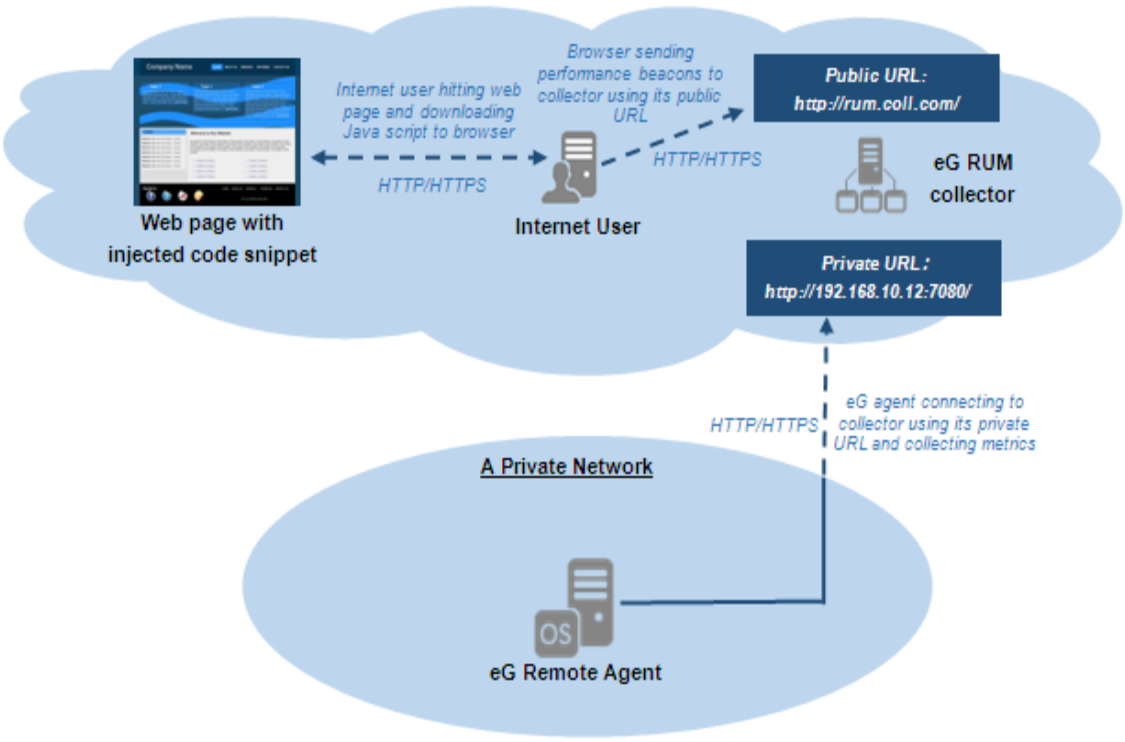


Figure 3.3: A RUM collector deployment model where the web application is used only by internet users and the eG agent alone is in a private network

If the web site/web application is being used by intranet users only, the eG RUM collector should also be deployed in the intranet (see Figure 3.4). If the eG agent is also deployed in the same network, then both the browser and the eG agent can communicate with the collector using its private URL.

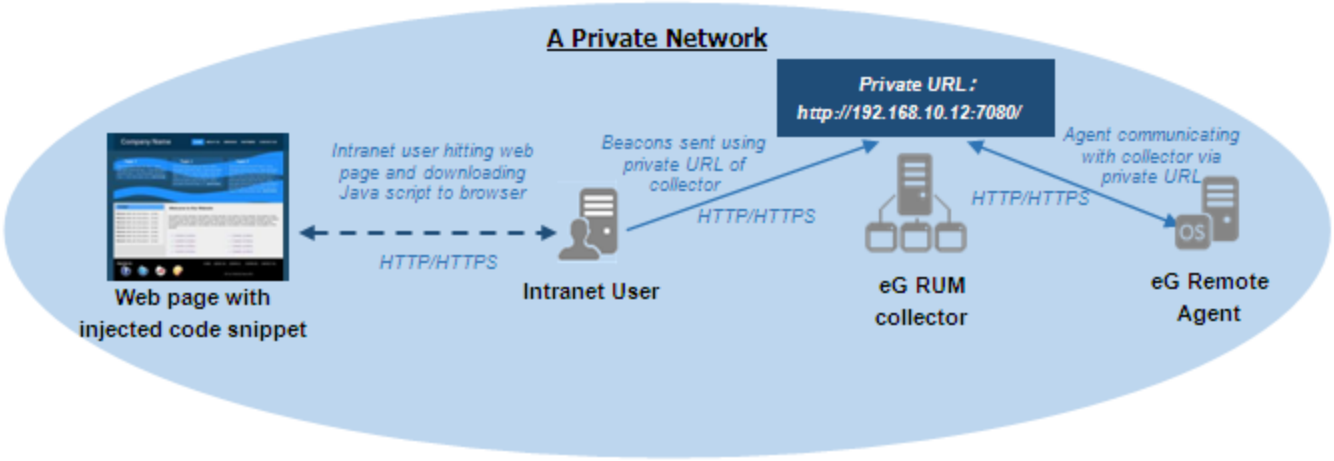


Figure 3.4: A RUM collector deployment model where the target web application is used only by intranet users and the eG agent is also in the same intranet

However, if in the above scenario, the eG agent is in a different sub-net or on the cloud, then the collector will have to be configured with both a private and a public IP address (see Figure 3.5).

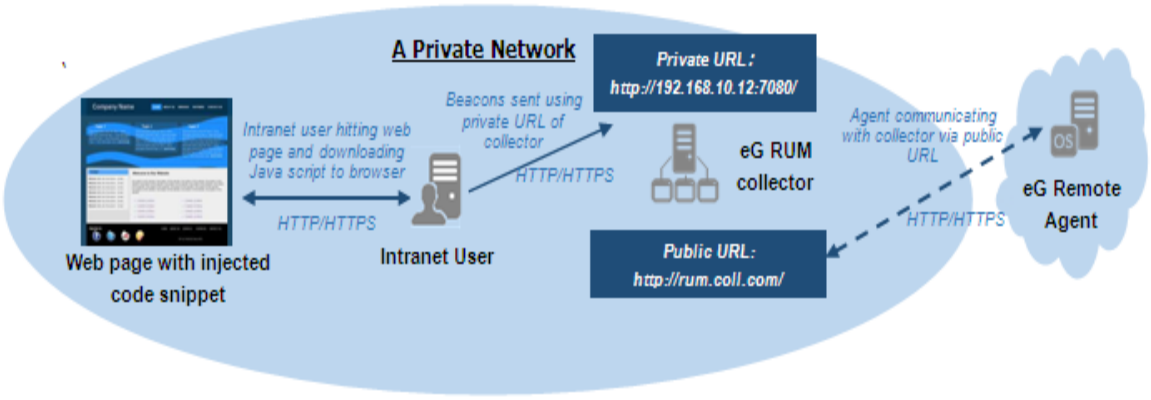


Figure 3.5: A RUM collector deployment model where the target web application is accessed by intranet users and the eG agent is on the cloud

3.2.3 Installing the eG RUM Collector

3.2.3.1 Installing the eG RUM Collector on Windows

The eG RUM collector for Windows is provided as a self-extracting program named **eGRUM**. You will have to execute this program on a Windows 2008 (64-bit) or Windows 2012 (64-bit) host in the environment to begin the installation. For this, first double-click on **eGRUM** on the target Windows host. The resulting installation wizard will guide you through the setup process.

1. The **Welcome screen** appears first (see Figure 3.6).

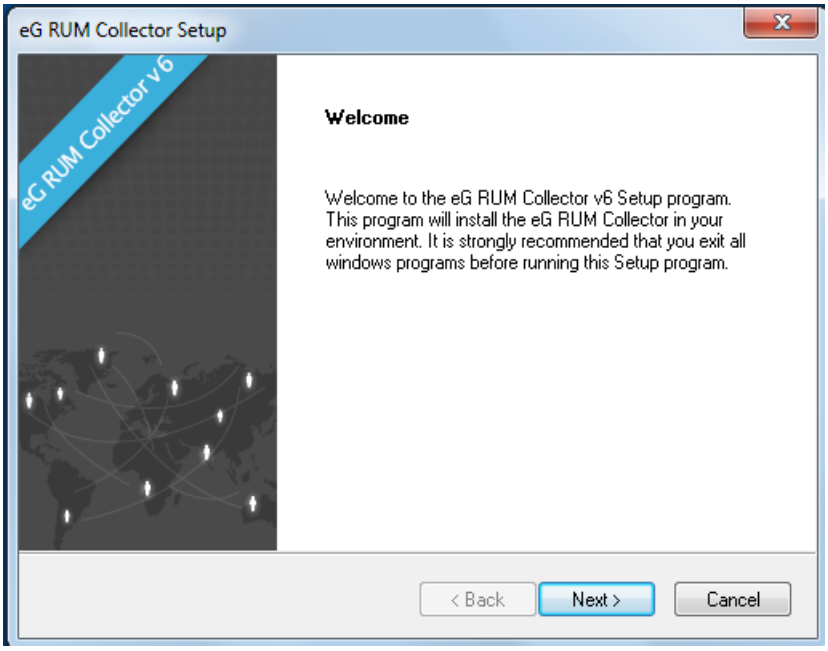


Figure 3.6: The Welcome screen

2. Click **Next** in Figure 3.6 to continue with the installation. Figure 3.7 will then appear.

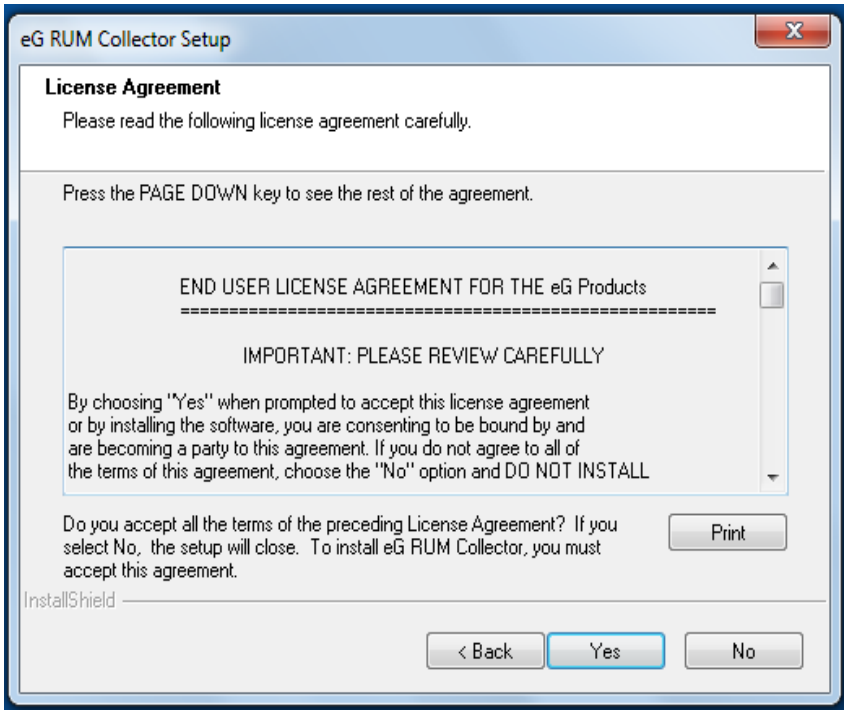


Figure 3.7: The End User License Agreement

3. Click the **Yes** button in Figure 3.7 to accept the terms of the license agreement. This will invoke Figure 3.8.

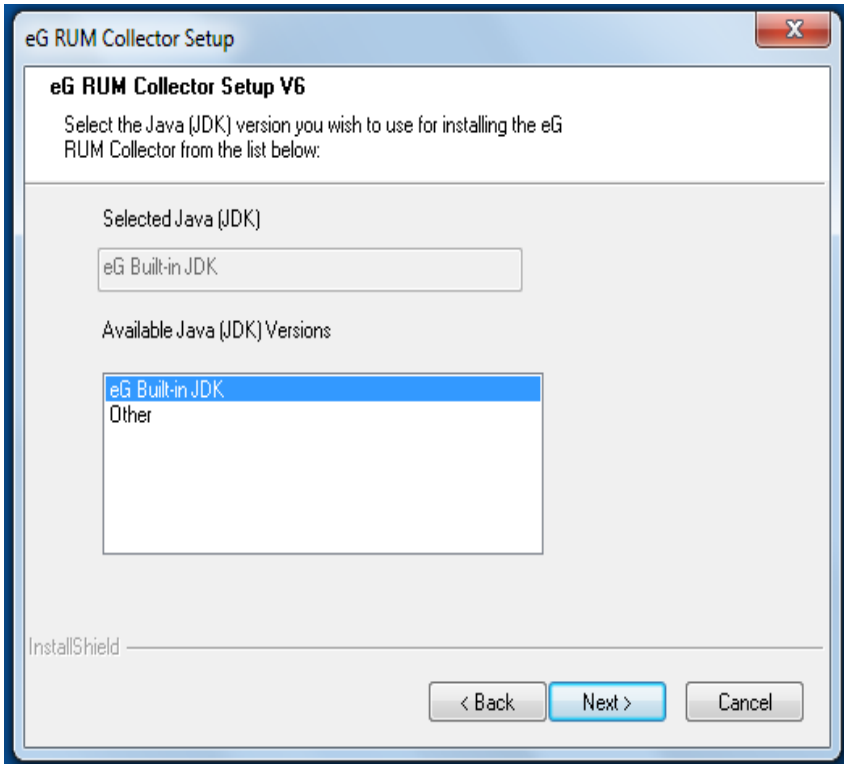


Figure 3.8: Selecting a JDK version to use for collector installation

4. Setup then automatically discovers all the versions of JDK available on the target host, and lists them as depicted by Figure 3.8. The user will have to simply select the JDK version he/she wants to use for their collector installation from the displayed list. **It is recommended that you use JDK 1.6 (or its variants) for installing the eG RUM collector**.

5. Upon selecting a JDK version, eG Enterprise automatically determines the location of the JDK-related files on the target host, and uses them to configure the eG user's Java execution environment to execute Java programs to proceed with the installation. Clicking on the **Next** button in Figure 3.8 will then lead the user straight to step 8 of the setup process.

6. On the other hand, if the JDK version the user wishes to use is not listed in Figure 3.8 for some reason, he/she can pick the **Other** option from Figure 3.8. Figure 3.9 then appears, where the user is prompted to specify if his/her environment contains the required JDK. **It is recommended that you use JDK 1.6 (or its variants) for installing the eG RUM collector.**

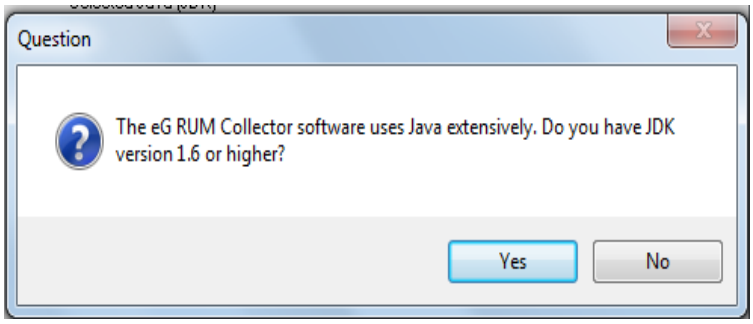


Figure 3.9: Setup enquiring the availability of JDK in the environment

7. If JDK is already available in the environment, specify the Java home directory to enable the setup process to configure the eG user's execution environment to execute Java programs as in Figure 3.10. The user can also use the **Browse** button to select the location of the Java home directory.

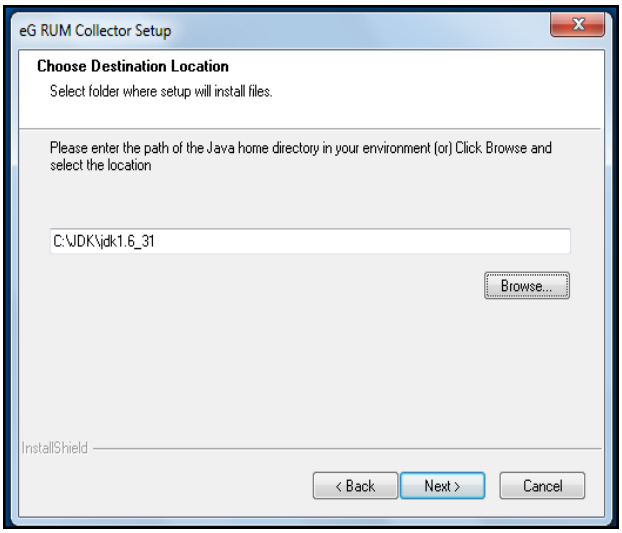


Figure 3.10: Specifying the location of the Java home directory for installing the eG RUM collector

8. The setup process now requires the hostname and port number of the host on which the eG RUM collector is being configured (see Figure 3.11). By default, setup auto-discovers the host name and the IP address (es) of the eG RUM collector, and makes it available for selection in Figure 3.11. You can pick the host name or any of the IP addresses listed therein to take the installation forward. If the IP address/host name that you want to use for your collector is not discovered for some reason, then, you can choose the **Other** option in Figure 3.11. This will invoke Figure 3.12 where you can manually specifiy the IP address/host name of the eG RUM collector. If the domain name service is used in the target environment, use the full hostname. Otherwise, specify the IP address. However, 7080 is the default port. You can change this port if you so need.

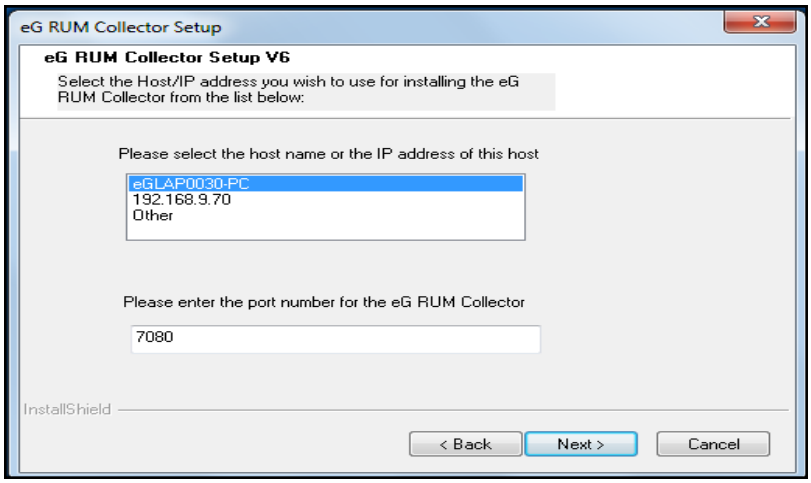


Figure 3.11: Selecting the IP address/host name of the collector

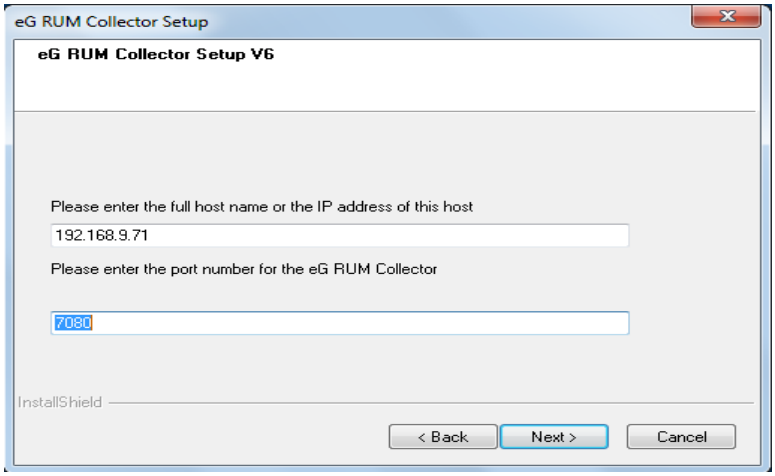


Figure 3.12: Manually specifying the IP address/host name of the collector host

Note:

- While specifying the host name (and not the IP address) of the collector, please take care of the following aspects:
  a. Use this name (and not the IP address) when configuring the eG RUM collector in the eG administrative interface.
  b. Make sure that forward and reverse lookups for this name are enabled via the DNS service in the target environment.
- When providing an IP address for the eG RUM collector, note that only an IPv4 address can be provided. To configure the eG collector on a host that has an IPv6 address, you will have to provide the fully-qualified host name of that host or an alias name, in Figure 3.12.

9. Setup then prompts you to indicate if the eG RUM collector is to be SSL-enabled. If so, click **Yes** in Figure 3.13. If not, click **No**.

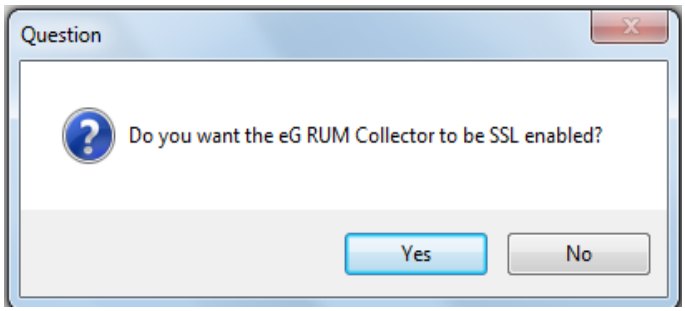


Figure 3.13: Indicating whether/not to SSL-enable the eG RUM collector

10. Next, use the **Browse** button in Figure 3.14 that appears to specify where the eG RUM collector is to be installed. Then, click **Next** to proceed.

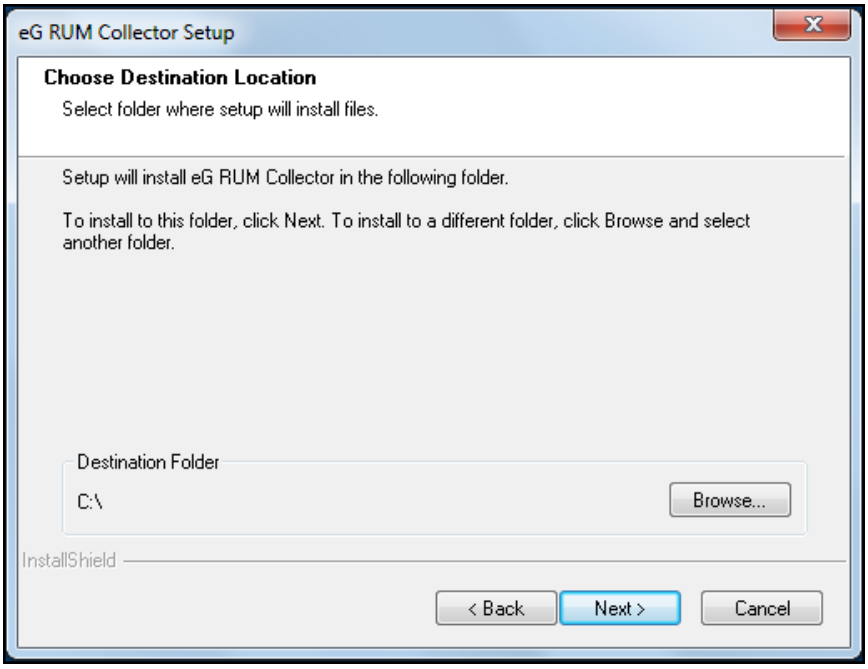


Figure 3.14: Specifying the install directory of the eG RUM collector

11. Then, view a quick summary of the specifications and click **Next** in Figure 3.15 to proceed.

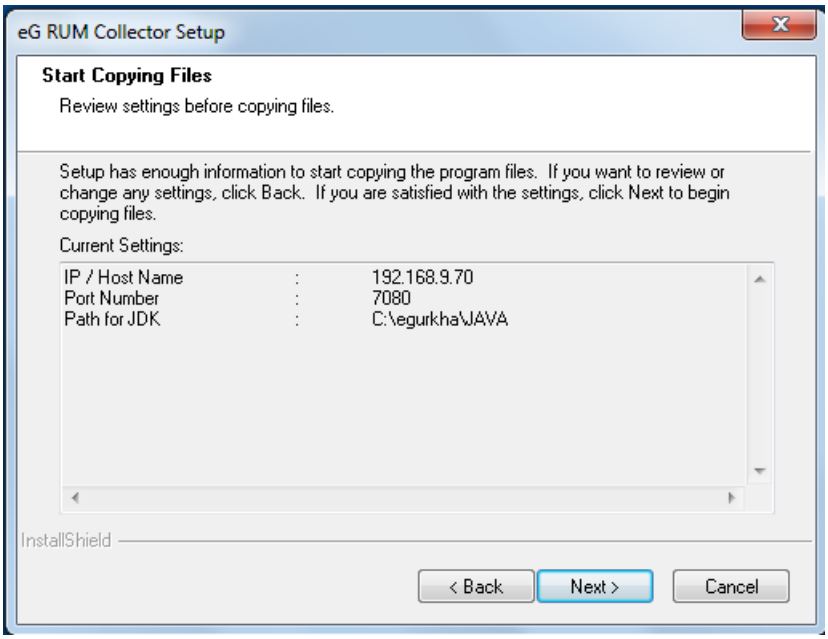


Figure 3.15: A summary of the specifications

12. If the configuration process succeeds, the following screen will be displayed (see Figure 3.16). The Setup requires the user to restart the system. This can be done immediately or at a later point of time. Clicking on the **Finish** button will exit the Setup.

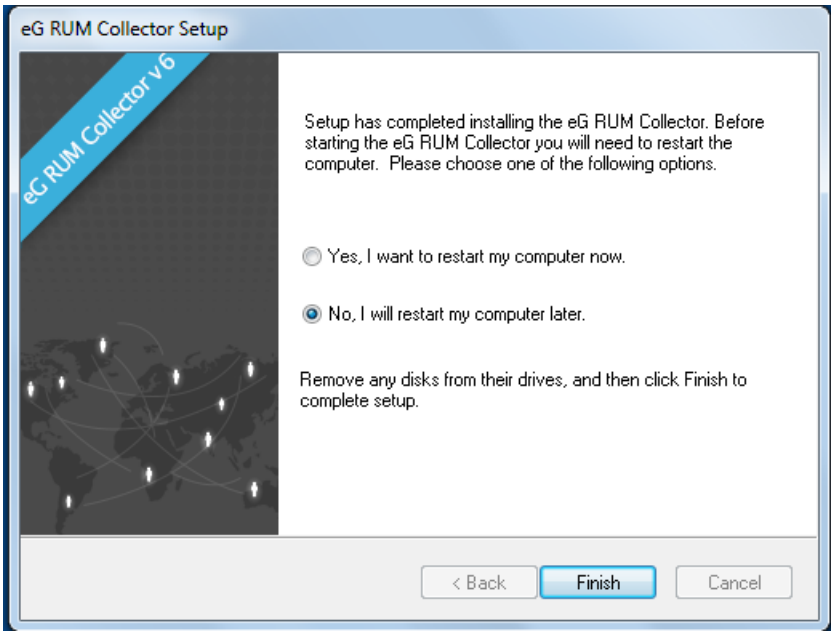


Figure 3.16: Setup program indicating the completion of the eG RUM collector installation

3.2.3.2 Installing the eG RUM Collector on Unix

The key pre-requisite for installing an eG RUM collector on Unix is to make sure that Apache Tomcat v6 (or above) pre-exists on the target host. If not, install Tomcat first and then proceed with the steps described in this section.

The eG RUM collector for Unix is available as a war file named **rumcollector.war** in the **/opt/egurkha/manager/tomcat/webapps** folder of the eG manager. To install the RUM collector on a different host, follow the steps below:

1. Copy the **rumcollector.war** file from the eG manager host to any location on the target RUM collector host.
2. Stop Tomcat on the server where the RUM collector is to be installed.
3. Open the shell prompt on the target host, and switch to the **<TOMCAT_HOME>/webapps** directory on the host.
4. Run the following command to copy the **rumcollector.war** file available in a temporary location on the host to the **<TOMCAT_HOME>/webapps:**

   ```
   cp /tmp/rumcollector.war
   ```

   Here */tmp/rumcollector.war* refers to the temporary location of the **rumcollector.war**.
5. Next, start the Tomcat server by running the following command from the **<TOMCAT_HOME>/bin** directory:

   ```
   catalina.sh run
   ```
6. Starting the Tomcat server will automatically start the **rumcollector** application deployed within. In the startup messages, look for the following line to confirm deployment:

   ```
   INFO: Deploying web application archive rumcollector.war
   ```
7. Once Tomcat starts, switch to the **<TOMCAT_HOME>/webapps** folder from the shell prompt, and run the following command to view the contents of the folder:

   ```
   ls -alt
   ```
8. Look for the **rumcollector** application file in the folder. If you find it, it is the first indication that the RUM collector has been successfully deployed on the target.
9. Next, from a remote host in the environment, launch a browser and connect to the following URL:

   http://<Tomcat_IP>:<Tomcat_connector_port>/rumcollector/Welcome.jsp
10. If this URL returns a page with the text *Welcome!* , it indicates the successful deployment of the RUM collector.

3.2.4 Starting/Stopping the RUM Collector

To start the RUM collector installed on a Windows host, login to the host, and follow the menu sequence: Start -> Programs -> eG Monitoring Suite -> Start RUM Collector.

If the collector is started properly on the Windows host, the following services will be started automatically on the host:

- A Tomcat service named, **eGRUM**
- The RUM collector's self-monitoring and recovery service named, **eGRumMon**

At any given point in point in time, you can stop the **eGRUM** service from the **Services** window to stop the RUM collector. Alternatively, you can use the menu sequence, Start -> Programs -> eG Monitoring Suite -> Stop RUM Collector, to stop the collector.

3.2.5 SSL-enabling the eG RUM Collector

To monitor the user experience with SSL-enabled web sites/web applications, it is **mandatory that you SSL-enable the eG RUM collector**. This is because, **an HTTP collector cannot manage an HTTPS web site/web application.**

Note:

An HTTPS collector on the other hand, can manage both HTTP and HTTPS web sites/web applications.

The eG RUM collector on Windows is bundled with a self-signed SSL certificate. This certificate is named **eGRUM.bin** and is available in the **<EG_RUM_COLLECTOR_INSTALL_DIR>\tomcat\webapps** folder. To SSL-enable the RUM collector on Windows using this certificate, all you need to do is click the **Yes** button (in Figure 3.13) at step 9 of the collector installation process detailed in Section **3.2.3.1** section of this document. **However, it is recommended that you do not use this self-signed certificate for SSL-enabling your Windows collectors.**

To SSL-enable any RUM collector – be it on Windows or Unix – you are advised to pick one of the two approaches outlined below:

- You can obtain a signed certificate from an internal certifiying authority (eg., Microsoft Active Directory Certificate Services) and use this certificate to SSL-enable the eG RUM collector, (OR)
- You can obtain a signed certificate from a valid, external certifying authority (eg., Verisign) and use this certificate to SSL-enable the eG RUM collector

If you go with option (a), use the procedure detailed in Section **3.2.5.1** section. If you pick option (b), use the procedure detailed in Section **3.2.5.2** section.

3.2.5.1 SSL-enabling the eG RUM Collector Using a Certificate Signed by an Internal CA

The broad steps to be followed to achieve this are as follows:

1. Generating the keystore file
2. Generating a certificate request
3. Submitting the certificate request to the internal Certificate Authority (CA) and obtaining a certificate

4. Importing the certificate into a keystore
5. Configuring Tomcat for using the keystore file

The sub-sections below elaborate on each of these steps.

The steps detailed below are common to both Windows and Unix installations of the collector.

3.2.5.1.1 Generating a Keystore File

The keystore file stores the details of the **certificates** necessary to make the protocol secure. Certificates contain the information pertaining to the source of the application data, and helps validate the source. To generate the keystore, use the **keytool** command. For this purpose, login to the RUM collector host and go to the command prompt. Set the **JAVA_HOME** path if it is not done already. Then, execute the following commands, one after another:

cd %JAVA_HOME%\bin (on Windows; on Unix, this will be **cd $JAVA_HOME/bin**)

keytool -genkey -alias **egitlab1** *-keyalg* **RSA** *-keypass* **mykey** *-keystore* **<Filename>***.keystore -storepass* **mykey** *-keysize* **2048** *-validity* **1095**

The text in **Bold** in the above command line indicates those inputs that can change according to the requirements of your environment. These inputs have been described below:

- **-alias** : an alias name for the certificate being generated
- **-keypass** : a password used to protect the key that is generated; **ensure that you provide the same values for -keypass and -storepass.**
- **-keyalg** : specifies the algorithm that is used to generate the keys. The options are as follows:
  - **DSA : Digital Signature Algorithm**
  - **RSA : An algorithm used for publick-key cryptography**
- **-keystore** : the *keytool* command stores the generated key in a *.keystore* file; provide a name for this file as input to the **-keystore** command
- **-keysize** : the size of the key that is generated; the default key size is 1024 bits - the key size must be in the range 512 bits - 1024 bits
- **-validity** : indicates the number of days for which the key/certicate will be valid - 1095 days refer to 3 years.

The command, upon execution, will request the following inputs:

```
What is your first and last name?
[Unknown]: <Type the eG RUM Collector's fully qualified domain name here>
What is the name of your organizational unit?
[Unknown]: United States
What is the name of your organization?
[Unknown]: eG Innovations Inc
What is the name of your City or Locality?
[Unknown]: Bridgewater
What is the name of your State or Province?
[Unknown]: New Jersey
What is the two-letter country code for this unit?
[Unknown]: US
Is CN=eG Innovations Inc, OU=United States, O=eG Innovations Inc, L=Bridge Water, ST=New
Jersey, C=US correct?
[no]: yes
```

When requested for the **first and last name**, indicate the *fully qualified domain name* using which you will be configuring the eG RUM collector in the eG admin interface. For instance, if the eG RUM collector is to be accessed using the URL, *http://egrumcollector.eginnovations.com*, where *egrumcollector.eginnovations.com* is the fully qualified domain name of the collector, then specify here.

Once all the required inputs are provided, a .keystore file will be generated in the **<JAVA_HOME_DIR>\bin** directory with the **<Filename>** you had provided while issuing the command.

3.2.5.1.2 Generating a Certificate Request

Once a keystore file is generated, proceed to request for a certificate from a valid certifying authority. The procedure for this is as follows:

1. Login to the eG RUM collector and go to its shell/command prompt.
2. Set the **JAVA_HOME** path if it is not done already.
3. Execute the following commands one after another:

   cd %JAVA_HOME%\bin (on Windows; on Unix, this will be **cd $JAVA_HOME/bin**)

   keytool -certreq -alias **egitlab1** *-keyalg* **RSA** *-file* **<Name_of_the_text_file>** *-keypass* **mykey** *-keystore* **<filename>.keystore** *–storepass* **mykey**

   The text in **Bold** in the above command line indicates those inputs that can change according to the requirements of your environment. These inputs have been described below:

   - **-alias** : the alias name of the certificate being requested; make sure that you provide the same alias name that you provided while generating the keystore file (see 3.2.5.1.1 section of this document).
   - **-keyalg** : specifies the algorithm that was used to generate the keys; this can be **RSA** or **DSA**, depending upon which algorithm was used for key generation in the procedure detailed in 3.2.5.1.1 section.
   - **-file** : Provide a name for the text file to which the certificate request will be saved.

- **-keypass** : the password used to protect the key that was generated; make sure that you provide the same password that you provided while generating the keystore file (see Section **3.2.5.1.1** section of this document). Also, note that **-storepass** and **-keypass** should be the same.
- **-keystore** : Provide the name of the *keystore* file in which the key has been stored; specify the same file name that you used to store the key (see Section **3.2.5.1.1** section of this document).

4. If this command executes successfully, then a certificate request will be generated and automatically stored in the text file you specified in step 2 above.

3.2.5.1.3 Obtaining a Certificate from the Internal CA

1. The first step towards obtaining a certificate is to submit the certificate request to the internal CA. For this connect to the Certificate server of the internal CA and select the option to submit the certificate. For instance, if you are using Microsoft Active Directory Certificate Services to request for a self-signed certificate, then, you need to connect to **http://<YourWebServerName>/certsrv**, and then pick the option to submit the certificate. Figure 3.17 will then appear.

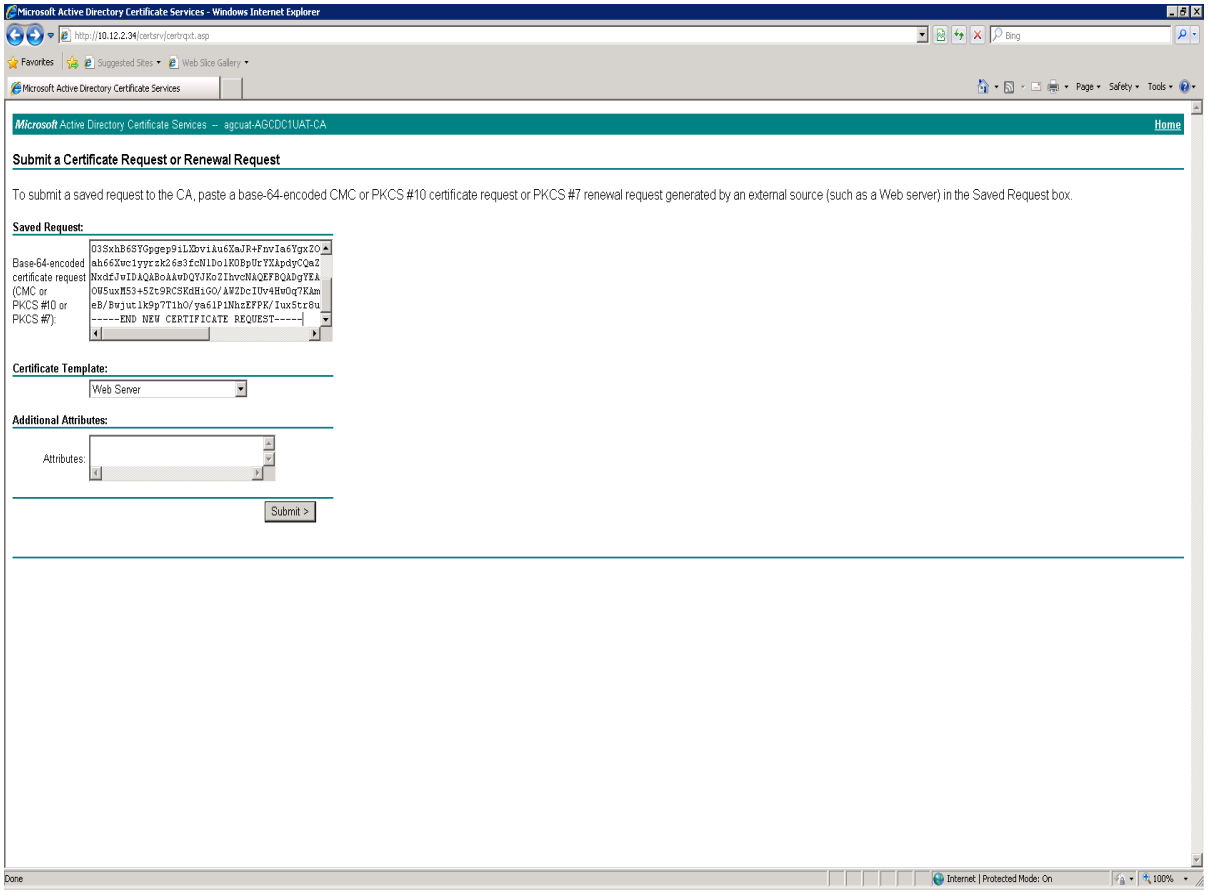


Figure 3.17: Requesting for a certificate

2. Open the text file containing the certificate request (which was created using the procedure detailed in Section **3.2.5.1.2** section above), copy the contents of the file, and then paste it to the text area of the **Base 64-encoded certificate request** text box of Figure 3.17. Then, click the **Submit** button.

3. The certificate will thus be generated. Download the certificate.

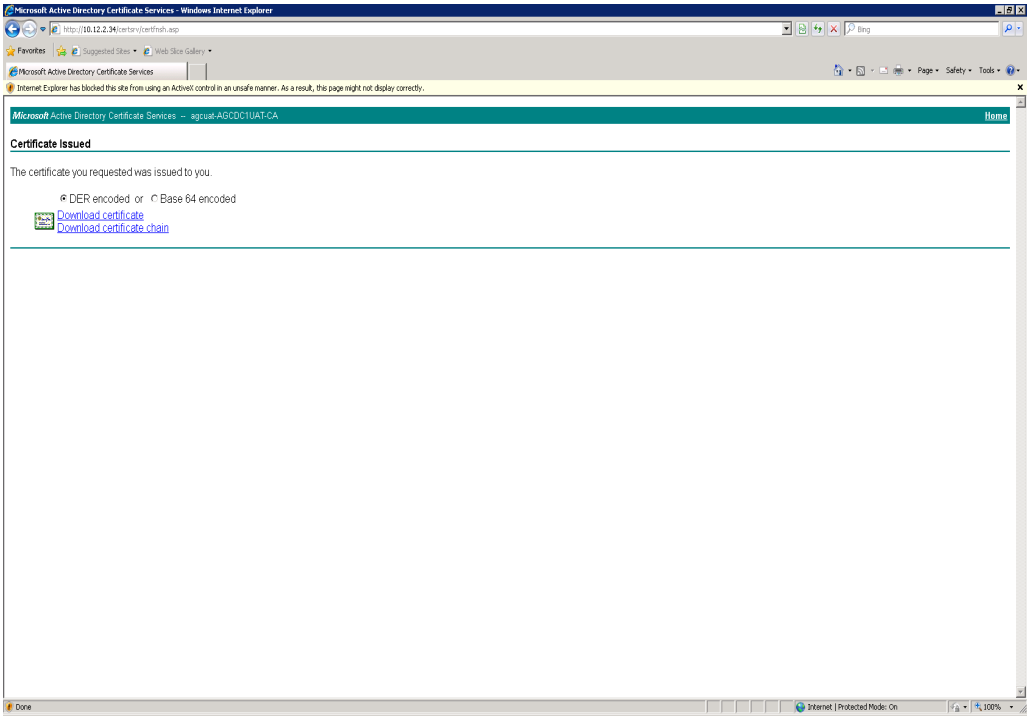


Figure 3.18: Downloading the certificate

3.2.5.1.4 Importing the Certificates into the Keystore File

The downloaded certificate can be in one of the following forms:

- Can be a single, combined certificate
- Can be accompanied by a certificate chain
- Can be in a PEM format

The procedure for importing certificates differs based on the format of the downloaded certificate. These procedures have been detailed in the sub-sections below.

3.2.5.1.5 Importing a Combined Certificate into the Keystore File

In this case, follow the steps below to import the certificate into the keystore file:

1. Set the **JAVA_HOME** path if it is not done already.
2. At the command prompt, execute the following commands, one after another:

   cd %JAVA_HOME%\bin (on Windows; on Unix, this will be **cd $JAVA_HOME/bin**)

   keytool -import –trustcacerts -alias **egitlab1** *-file* **<Name_of_the_domain_certificate>** *-keystore* **<Name_of_the_keystore_file>.keystore**

   The text in **Bold** in the above command line indicates those inputs that can change according to the requirements of your environment. These inputs have been described below:

- **-alias** : the alias name of the certificate being requested; make sure that you provide the same alias name you provided when generating the keystore (see Section **3.2.5.1.1**)
- **-file** : the name of the domain certificate that you want to import
- **-keystore** : Provide the name of the *keystore* file you generated when you followed the procedure detailed in Section Section **3.2.5.1.1** above.

3.2.5.1.6 Importing a Signed Certificate and the Certificate Chain into the Keystore File

Digital certificates are verified using a chain of trust. The trust anchor for the digital certificate is the Root Certificate Authority (CA). The Certificate Hierarchy is a structure of certificates that allows individuals to verify the validity of a certificate's issuer. Certificates are issued and signed by certificates that reside higher in the certificate hierarchy, so the validity and trustworthiness of a given certificate is determined by the corresponding validity of the certificate that signed it.

The Chain of Trust of a Certificate Chain is an ordered list of certificates, containing an end-user subscriber certificate and intermediate certificates (that represents the Intermediate CA), that enables the receiver to verify that the sender and all intermediate certificates are trustworthy.

A certificate chain will therefore consist of multiple certificates. Before importing each of these certificates, **you will have to understand the hierarchy of the certificates**. To know which is the root and which is the intermediate certificate, refer to the web site of the certificate authority. Then, set the **JAVA_HOME** path if it is not done already.

Then, follow the steps below:

1. First, import the Root certificate. For this, execute the following commands, one after another in the command prompt:

   cd %JAVA_HOME%\bin (on Windows; on Unix, this will be **cd $JAVA_HOME/bin**)

   keytool -import –trustcacerts -alias **rootcert** *-file* **<Name_of_the_root_certificate>** *-keystore* **<Name_of_the_keystore_file>.keystore** *–keypass* **mykey** *–storepass* **mykey**

   The text in **Bold** in the above command line indicates those inputs that can change according to the requirements of your environment. These inputs have been described below:

   - **-alias** : the alias name of the certificate being requested; make sure that you provide a unique alias name for the root certificate.
   - **-file**: the name of the root certificate that you want to import
   - **-keystore** : Provide the name of the *keystore* file you generated when you followed the procedure detailed in Section **3.2.5.1.1** section.
   - **-keypass** and **–storepass** : Provide the same **keypass** and **storepass** that you specified when generating the keystore; refer to Section **3.2.5.1.1** section above for details.

2. Next, import each of the intermediate certificates, one after another, using the following command:
   keytool -import –trustcacerts -alias **intercert1***-file* **<Name_of_the_intermediate_certificate>***-keystore* **<Name_of_the_keystore_file>.keystore***–keypass***mykey***–storepass***mykey**

   The text in **Bold** in the above command line indicates those inputs that can change according to the requirements of your environment. These inputs have been described below:

   - **-alias** : the alias name of the certificate being requested; make sure that you provide a unique alias name for every intermediate certificate.
   - **-file**: the name of the intermediate certificate that you want to import
   - **-keystore** : Provide the name of the *keystore* file you generated when you followed the procedure detailed in Section **3.2.5.1.1** section.
   - **-keypass** and **–storepass** : Provide the same **keypass** and **storepass** that you specified when generating the keystore; refer to Section Section **3.2.5.1.1** above for details.

3. Finally, import the entity/domain certificate into the keystore by issuing the following command:
   keytool -import –trustcacerts -alias **egitlab1** *-file* **<Name_of_the_domain_certificate>** *-keystore* **<Name_of_the_keystore_file>.keystore**

   The text in **Bold** in the above command line indicates those inputs that can change according to the requirements of your environment. These inputs have been described below:

   - **-alias** : the alias name of the certificate being requested; make sure that you provide the same alias name you provided when generating the keystore (see Section **3.2.5.1.1**) .
   - **-file**: the name of the domain certificate that you want to import
   - **-keystore** : Provide the name of the *keystore* file you generated when you followed the procedure detailed in Section **3.2.5.1.1** .

Note:

If the domain certificate import command throws an error for any reason, it could be because, all related certificates may not have been imported. Check the web site of the CA for more details.

3.2.5.1.7 Importing a Certificate that is in the PEM Format

PEM is a container format that may include just the public certificate (such as with Apache installs, and CA certificate files */etc/ssl/certs*), or may include an entire certificate chain including public key, private key, and root certificates, or may only contain a certificate and a private key.

If the certificate you downloaded is in the PEM format and includes only a certificate file and a private key file, then follow the steps below to import that certificate into a keystore file.

1. Run the following command from the command prompt to export the certificate and private key file into the pkcs12 format:

   openssl pkcs12 -export -in **certificate.crt** *-inkey* **private.key** *-certfile* **certificate.crt** *-name "***My certificate"** *-out* **keystore***.p12*

The text in **Bold** in the above command line indicates those inputs that can change according to the requirements of your environment. These inputs have been described below:

- **-in** : the name of the certificate that is included in the PEM container
- **-inkey** : the name of the private key file the PEM container consists of
- **-certfile** : the name of the certificate that is included in the PEM container
- **-name** : Provide a **unique name for the certificate file** that is being exported.
- -**out :** Specify the name of the keystore file to which the certificate and private key are to be exported. **The keystore file can have any name of your choice.**

2. Next, you need to convert the keystore file, which is currently in the pkcs12 format, into the Java keystore (i.e., JKS) format. For this, issue the following command at the command prompt:

   keytool -importkeystore –alias **egitlab1** *-deststorepass* **mykey** *-destkeypass* **mykey** *-destkeystore* **keystore,jks** *-srckeystore* **keystore.pk12** *-srcstoretype PKCS12 -srcstorepass* **mykey**

   The text in **Bold** in the above command line indicates those inputs that can change according to the requirements of your environment. These inputs have been described below:

   - **-alias** : the aliasname of the certificate being requested; make sure that you provide the same alias name that you specified in Section **3.2.5.1.1** of this document.
   - **-deststorepass** : this refers to the **storepass** of the destination keystore file – i.e., the keystore file in the JKS format. **The storepass of the destination keystore should be the same as the storepass of the source keystore.**
   - **-destkeypass** : this refers to the **keypass** of the destination keystore file - i.e., the keystore file in the JKS format. **The storepass and keypass of the destination keystore file should be the same.**
   - **-destkeystore**: the name of the destination keystore file – i.e., the keystore file in the JKS format.
   - **-srckeystore** : the name of the destination keystore file – i.e., the keystore file in the PKCS12 format.
   - **-srcstorepass** : The **storepass** of the source keystore file – i.e., the keystore file in the PKCS12 firmat. make sure that you provide the same storepass you specified in Section **3.2.5.1.1** section of this document

3.2.5.1.8 Configuring Tomcat for Using the Keystore File

The eG RUM collector is a Tomcat-based software. Therefore, to SSL-enable the eG RUM collector, you need to configure the **server.xml** file of Tomcat with the name and full path to the keystore file which was created earlier.

1. Edit the **server.xml** file in the **<CATALINA_HOME>\conf** directory.

2. In the file, search for the XML block where the SSL Coyote HTTP connector on port 8443 is defined. If this block is commented, it indicates that the eG RUM collector is not SSL-enabled and is hence listening on an HTTP port only. To SSL-enable the eG RUM collector, first uncomment this block as indicated below:

```
<!- -  Define  a  SSL  Coyote  HTTP/1.1  Connector  on  port  8443  - - >
                          <Connector                          protocol="HTTP/1.1"
               port="8443"          minSpareThreads="32"          maxThreads="512"
          enableLookups="false"     acceptCount="10"      connectionTimeout="20000"
          useURIValidationHack="false"    URIEncoding="UTF-  8"    compression="on"
compressionMinSize="2048"         noCompressionUserAgents="gozilla,         traviata"
compressableMimeType="text/html,text/xml,text/plain,application/x-              java-
applet,application/octet-
stream,application/xml,text/javascript,text/css,image/png,image/jpeg,image/gif,applica
tion/pdf,
application/x-
javascript,application/javascript"
                SSLEnabled="true"           scheme="https"           secure="true"
      clientAuth="false" sslProtocol="TLS"/>
```

3. Then, proceed to make the changes indicated in **Bold** below in the SSL XML block:

```
<!- -  Define  a  SSL  Coyote  HTTP/1.1  Connector  on  port  8443  - - >
                          <Connector                          protocol="HTTP/1.1"
            port="  7077  "        minSpareThreads="32"        maxThreads="512"
          enableLookups="false"     acceptCount="10"      connectionTimeout="20000"
useURIValidationHack="false"        URIEncoding="UTF-    8"        compression="on"
compressionMinSize="2048"         noCompressionUserAgents="gozilla,         traviata"
compressableMimeType="text/html,text/xml,text/plain,application/x-              java-
applet,application/octet-
stream,application/xml,text/javascript,text/css,image/png,image/jpeg,image/gif,
application/pdf,application/x-                    javascript,application/javascript"
                SSLEnabled="true"           scheme="https"           secure="true"
      clientAuth="false" sslProtocol="TLS" keyAlias=”egitlab1” keystoreFile="<The_full_
path_to_the_keystore_file>"keystorePass="mykey" />
```

Set the *port* parameter in the XML block to reflect the SSL port number that you have configured for the eG RUM collector. Also, note that three new parameters, namely - **keyAlias, keystoreFile** and **keystorePass** - have been inserted into the SSL block. While the **keystoreFile** parameter has to be set to the full path to the **.keystore** file that you generated earlier, the **keystorePass** parameter should be set to the keystore password that you specified while issuing the **keytool** command. Likewise, the **keyAlias** parameter is to be set to the **alias name** that you provided for the certificate file, when you generated it in Section **3.2.5.1.1** section above.

4. With that change, the eG RUM collector has acquired the capability to listen on two ports - the SSL port and the non-SSL port. To configure the eG RUM collector to listen only on the SSL port, simply comment that section of the **server.xml** file where the non-SSL Coyote HTTP connector on port 8081 has been defined, as indicated below:

```
<!-- Define a non-SSL Coyote HTTP/1.1 Connector on port 8081 -->
    <!-- <Connector className="org.apache.coyote.tomcat4.CoyoteConnector"
   Port7077" minSpareThreads="32" maxThreads="512"
               enableLookups="true" redirectPort="8443"
               acceptCount="10" debug="0" connectionTimeout="20000"
               useURIValidationHack="false"  URIEncoding="UTF-8"/> -->
```

5. Save the file.

6. Finally, start the eG RUM collector.

3.2.5.2 SSL-Enabling the eG RUM Collector Using a Signed Certificate Obtained from a Valid Certifying Authority

In highly secure environments, especially where the eG RUM collector is to be frequently accessed via the public internet, using a certificate signed by an internal CA may not be preferred. In such a case, you can obtain a valid certificate from a certificate authority and use that certificate to SSL-enable the eG RUM collector.

The broad steps to be followed to achieve this are as follows:

1. Generating the keystore file

2. Generating a certificate request

3. Submitting the certificate request to the Certificate Authority (CA) and obtaining a certificate

4. Importing the certificate into a keystore

5. Configuring Tomcat for using the keystore file

The sub-sections below elaborate on each of these steps.

The steps detailed below are common to both Windows and Unix installations of the collector.

3.2.5.2.1 Generating a Keystore File

The keystore file stores the details of the **certificates** necessary to make the protocol secure. Certificates contain the information pertaining to the source of the application data, and helps validate the source. To generate the keystore, use the **keytool** command. For this purpose, login to the RUM collector and go to the Unix shell/Windows command prompt (as the case may be). Set the **JAVA_HOME** path if it is not done already. Then, execute the following commands, one after another:

cd %JAVA_HOME%\bin (on Windows; on Unix, this will be **cd $JAVA_HOME/bin**)

keytool -genkey -alias **egitlab1** *-keyalg* **RSA** **-***keypass* **mykey** *-keystore* **<Filename>***.keystore -storepass* **mykey** *-keysize* **2048** *-validity* **1095**

The text in **Bold** in the above command line indicates those inputs that can change according to the requirements of your environment. These inputs have been described below:

- **-alias** : an alias name for the certificate being generated
- -**keypass** : a password used to protect the key that is generated; **ensure that you provide the same values for -keypass and -storepass.**
- **-keyalg** : specifies the algorithm that is used to generate the keys. The options are as follows:
  - **DSA**: Digital Signature Algorithm
  - **RSA :** An algorithm used for publick-key cryptography
- **-keystore** : the *keytool* command stores the generated key in a *.keystore* file; provide a name for this file as input to the **-keystore** command
- **-keysize** : the size of the key that is generated; the default key size is 1024 bits - the key size must be in the range 512 bits - 1024 bits
- **-validity** : indicates the number of days for which the key/certicate will be valid - 1095 days refer to 3 years.

The command, upon execution, will request the following inputs:

```
What is your first and last name?
[Unknown]: <Type the eG manager's fully qualified domain name here>
What is the name of your organizational unit?
[Unknown]: United States
What is the name of your organization?
[Unknown]: eG Innovations Inc
What is the name of your City or Locality?
[Unknown]: Bridgewater
What is the name of your State or Province?
[Unknown]: New Jersey
What is the two-letter country code for this unit?
[Unknown]: US
Is CN=eG Innovations Inc, OU=United States, O=eG Innovations Inc, L=Bridge Water, ST=New
Jersey, C=US correct?
[no]: yes
```

When requested for the **first and last name**, indicate the *fully qualified domain name* using which you will be accessing the eG RUM collector. For instance, if the collector is to be accessed as *http://egrumcollector.eginnovations.com* , where *egrumcollector.eginnovations.com* is the fully qualified domain name of the RUM collector, then specify *egrumcollector.eginnovations.com* here.

Once all the required inputs are provided, a .keystore file will be generated in the **<JAVA_HOME_DIR>\bin** directory with the **<Filename>** you had provided while issuing the command.

3.2.5.2.2 Generating a Certificate Request

Once a keystore file is generated, proceed to request for a certificate from a valid certifying authority. The procedure for this is as follows:

1. Login to the eG RUM collector host and go to the Unix shell/Windows command prompt (as the case may be).
2. Set the **JAVA_HOME** path if it is not done already.
3. Execute the following commands one after another:

   cd %JAVA_HOME%\bin (on Windows; on Unix, this will be **cd $JAVA_HOME/bin**)

   keytool -certreq -alias **egitlab1** *-keyalg* **RSA** *-file* **<Name_of_the_text_file>** *-keypass* **mykey** *-keystore* **<filename>.keystore** *–storepass* **mykey**

   The text in **Bold** in the above command line indicates those inputs that can change according to the requirements of your environment**.** These inputs have been described below:

   - **-alias** : the alias name of the certificate being requested; make sure that you provide the same alias name that you provided while generating the keystore file (see Section **3.2.5.1.1** of this document).
   - **-keyalg** : specifies the algorithm that was used to generate the keys; this can be **RSA** or **DSA**, depending upon which algorithm was used for key generation in the procedure detailed in Section **3.2.5.1.1**
   - **-file** : Provide a name for the text file to which the certificate request will be saved.
   - -**keypass** : the password used to protect the key that was generated; make sure that you provide the same password that you provided while generating the keystore file (see Section **3.2.5.1.1** section of this document). Also, note that **-storepass** and **-keypass** should be the same.
   - **-keystore** : Provide the name of the *keystore* file in which the key has been stored; specify the same file name that you used to store the key (see Section **3.2.5.1.1** section of this document).

4. If this command executes successfully, then a certificate request will be generated and automatically stored in the text file you specified in step 2 above.

3.2.5.2.3 Obtaining the Certificate from the CA

1. The first step towards obtaining a certificate is to submit the certificate request to the CA. For this connect to the Certificate server of the CA and submit the certificate. The procedure for request submission will differ from one CA to another.

2. The certificate will thus be generated. Download the certificate.

3.2.5.2.4 Importing the Certificates into the Keystore File

The downloaded certificate can be in one of the following forms:

- Can be a single, combined certificate
- Can be accompanied by a certificate chain
- Can be in a PEM format

The procedure for importing certificates differs based on the format of the downloaded certificate. These procedures have been detailed in the sub-sections below.

3.2.5.2.5 Importing a Combined Certificate into the Keystore File

In this case, follow the steps below to import the certificate into the keystore file:

1. Set the **JAVA_HOME** path if it is not done already.
2. At the command prompt, execute the following commands, one after another:

   cd %JAVA_HOME%\bin (on Windows; on Unix, this will be **cd $JAVA_HOME/bin**)

   keytool -import –trustcacerts -alias **egitlab1** *-file* **<Name_of_the_domain_certificate>** *-keystore* **<Name_of_the_keystore_file>.keystore**

   The text in **Bold** in the above command line indicates those inputs that can change according to the requirements of your environment. These inputs have been described below:

   - **-alias** : the alias name of the certificate being requested; **make sure that you provide the same alias name you provided when generating the keystore (see** Section **3.2.5.1.1**) **.**
   - **-file**: the name of the domain certificate that you want to import
   - **-keystore** : Provide the name of the *keystore* file you generated when you followed the procedure detailed in Section **3.2.5.1.1** section above.

3.2.5.2.6 Importing a Signed Certificate and the Certificate Chain into the Keystore File

Digital certificates are verified using a chain of trust. The trust anchor for the digital certificate is the Root Certificate Authority (CA). The Certificate Hierarchy is a structure of certificates that allows individuals to verify the validity of a certificate's issuer. Certificates are issued and signed by certificates that reside higher in the certificate hierarchy, so the validity and trustworthiness of a given certificate is determined by the corresponding validity of the certificate that signed it.

The Chain of Trust of a Certificate Chain is an ordered list of certificates, containing an end-user subscriber certificate and intermediate certificates (that represents the Intermediate CA), that enables the receiver to verify that the sender and all intermediate certificates are trustworthy.

A certificate chain will therefore consist of multiple certificates. Before importing each of these certificates, **you will have to understand the hierarchy of the certificates**. To know which is the root and which is the intermediate certificate, refer to the web site of the certificate authority. For instance, if Comodo is the Certificate Authority that has issued the SSL certificate, then connect to the following URL, https://support.comodo.com/index.php?/Default/Knowledgebase/Article/View/620/1/, to gain clarity.

Then, follow the steps below:

1. Set the **JAVA_HOME** path if it is not done already.
2. Then, import the Root certificate. For this, execute the following commands, one after another in the command prompt:

   cd %JAVA_HOME%\bin (on Windows; on Unix, this will be **cd $JAVA_HOME/bin**)

   keytool -import –trustcacerts -alias **rootcert** *-file* **<Name_of_the_root_certificate>** *-keystore* **<Name_of_the_keystore_file>.keystore** *–keypass* **mykey** *–storepass* **mykey**

   The text in **Bold** in the above command line indicates those inputs that can change according to the requirements of your environment. These inputs have been described below:

   - **-alias** : the alias name of the certificate being requested; make sure that you provide a unique alias name for the root certificate.
   - **-file**: the name of the root certificate that you want to import
   - **-keystore** : Provide the name of the *keystore* file you generated when you followed the procedure detailed in Section Section **3.2.5.1.1** above.
   - **-keypass** and **–storepass** : Provide the same **keypass** and **storepass** that you specified when generating the keystore; refer to Section **3.2.5.1.1** section above for details.

3. Next, import each of the intermediate certificates, one after another, using the following command:
   keytool -import –trustcacerts -alias **intercert1***-file* **<Name_of_the_intermediate_certificate>***-keystore* **<Name_of_the_keystore_file>.keystore***–keypass***mykey***–storepass***mykey**

   The text in **Bold** in the above command line indicates those inputs that can change according to the requirements of your environment. These inputs have been described below:

   - **-alias** : the alias name of the certificate being requested; make sure that you provide a unique alias name for every intermediate certificate.
   - **-file**: the name of the intermediate certificate that you want to import
   - **-keystore** : Provide the name of the *keystore* file you generated when you followed the procedure detailed in Section Section **3.2.5.1.1** above.
   - **-keypass** and **–storepass** : Provide the same **keypass** and **storepass** that you specified when generating the keystore; refer to Section **3.2.5.1.1** above for details.

4. Finally, import the entity/domain certificate into the keystore by issuing the following command:
   keytool -import –trustcacerts -alias **egitlab1** *-file* **<Name_of_the_domain_certificate>** *-keystore* **<Name_of_the_keystore_file>.keystore**

   The text in **Bold** in the above command line indicates those inputs that can change according to the requirements of your environment. These inputs have been described below:

   - **-alias** : the alias name of the certificate being requested; **make sure that you provide the same alias name you provided when generating the keystore (see** Section **3.2.5.1.1**) **.**

- **-file**: the name of the domain certificate that you want to import
- **-keystore** : Provide the name of the *keystore* file you generated when you followed the procedure detailed in Section **3.2.5.1.1** section above.

Note:

If the domain certificate import command throws an error for any reason, it could be because, all related certificates may not have been imported. Check the web site of the CA for more details.

3.2.5.2.7 Importing a Certificate that is in the PEM Format

PEM is a container format that may include just the public certificate (such as with Apache installs, and CA certificate files */etc/ssl/certs*), or may include an entire certificate chain including public key, private key, and root certificates, or may only contain a certificate and a private key.

If the certificate you downloaded is in the PEM format and includes only a certificate file and a private key file, then follow the steps below to import that certificate into a keystore file.

1. Run the following command from the command prompt to export the certificate and private key file into the pkcs12 format:
   openssl pkcs12 -export -in **certificate.crt** *-inkey* **private.key** *-certfile* **certificate.crt** *-name* **"My certificate"** *-out* **keystore***.p12*

   The text in **Bold** in the above command line indicates those inputs that can change according to the requirements of your environment. These inputs have been described below:

   - **-in** : the name of the certificate that is included in the PEM container
   - **-inkey**: the name of the private key file the PEM container consists of
   - **-certfile** : the name of the certificate that is included in the PEM container
   - **-name** : Provide a **unique name for the certificate file** that is being exported.
   - -**out :** Specify the name of the keystore file to which the certificate and private key are to be exported. **The keystore file can have any name of your choice.**

2. Next, you need to convert the keystore file, which is currently in the pkcs12 format, into the Java keystore (i.e., JKS) format. For this, issue the following command at the command prompt:

   keytool -importkeystore –alias **egitlab1** *-deststorepass* **mykey** *-destkeypass* **mykey** *-destkeystore* **keystore,jks** *-srckeystore* **keystore.pk12** *-srcstoretype PKCS12 -srcstorepass* **mykey**

   The text in **Bold** in the above command line indicates those inputs that can change according to the requirements of your environment. These inputs have been described below:

   - **-alias** : the aliasname of the certificate being requested; make sure that you provide the same alias name that you specified in Section **3.2.5.1.1** section of this document.

- **-deststorepass** : this refers to the **storepass** of the destination keystore file – i.e., the keystore file in the JKS format. **The storepass of the destination keystore should be the same as the storepass of the source keystore.**
- **-destkeypass** : this refers to the **keypass** of the destination keystore file - i.e., the keystore file in the JKS format. **The storepass and keypass of the destination keystore file should be the same.**
- **-destkeystore**: the name of the destination keystore file – i.e., the keystore file in the JKS format.
- **-srckeystore** : the name of the destination keystore file – i.e., the keystore file in the PKCS12 format.
- **-srcstorepass** : The **storepass** of the source keystore file – i.e., the keystore file in the PKCS12 format. make sure that you provide the same storepass you specified in Section **3.2.5.1.1** section of this document

3.2.5.2.8 Configuring Tomcat for Using the Keystore File

The eG RUM collector uses Tomcat as the web server. Therefore, to SSL-enable the collector, you need to configure the **server.xml** file of Tomcat with the name and full path to the keystore file which was created earlier.

1. Edit the **server.xml** file in the **<CATALINA_HOME>\conf** directory.
2. In the file, search for the XML block where the SSL Coyote HTTP connector on port 8443 is defined. If this block is commented, it indicates that the eG collector is not SSL-enabled and is hence listening on an HTTP port only. To SSL-enable the eG RUM collector, first uncomment this block as indicated below:

```
<!-- Define a SSL Coyote HTTP/1.1 Connector on port 8443 -->
    <Connector protocol="HTTP/1.1"
       port="8443" minSpareThreads="32" maxThreads="512"
       enableLookups="false" acceptCount="10"  connectionTimeout="20000"
       useURIValidationHack="false" URIEncoding="UTF-8" compression="on"
compressionMinSize="2048" noCompressionUserAgents="gozilla, traviata"
compressableMimeType="text/html,text/xml,text/plain,application/x-java-
applet,application/octet-
stream,application/xml,text/javascript,text/css,image/png,image/jpeg,image/gif,applica
tion/pdf,application/
x-javascript,application/javascript"
       SSLEnabled="true" scheme="https" secure="true"
       clientAuth="false" sslProtocol="TLS"/>
```

3. Then, proceed to make the changes indicated in **Bold** below in the SSL XML block:

```
<!-- Define a SSL Coyote HTTP/1.1 Connector on port 8443 -->
    <Connector protocol="HTTP/1.1"
       port="7080" minSpareThreads="32" maxThreads="512"
       enableLookups="false" acceptCount="10"  connectionTimeout="20000"
useURIValidationHack="false" URIEncoding="UTF-8" compression="on"
compressionMinSize="2048" noCompressionUserAgents="gozilla, traviata"
compressableMimeType="text/html,text/xml,text/plain,application/x-java-
applet,application/octet-
stream,application/xml,text/javascript,text/css,image/png,image/jpeg,image/gif,
application/pdf,application/x-javascript,application/javascript"
       SSLEnabled="true" scheme="https" secure="true"
       clientAuth="false" sslProtocol="TLS" keyAlias=”egitlab1” keystoreFile="<The_full_
path_to_the_keystore_file>"keystorePass="mykey" />
```

Set the *port* parameter in the XML block to reflect the SSL port number that you have configured for the eG RUM collector. Also, note that three new parameters, namely - **keyAlias, keystoreFile** and **keystorePass** - have been inserted into the SSL block. While the **keystoreFile** parameter has to be set to the full path to the **.keystore** file that you generated earlier, the **keystorePass** parameter should be set to the keystore password that you specified while issuing the **keytool** command. Likewise, the **keyAlias** parameter is to be set to the **alias name** that you provided for the certificate file, when you generated it in Section **3.2.5.1.1** section above.

4. With that change, the eG RUM collector has acquired the capability to listen on two ports - the SSL port and the non-SSL port. To configure the collector to listen only on the SSL port, simply comment that section of the **server.xml** file where the non-SSL Coyote HTTP connector on port 8081 has been defined, as indicated below:

```
<!-- Define a non-SSL Coyote HTTP/1.1 Connector on port 8081 -->
    <!-- <Connector className="org.apache.coyote.tomcat4.CoyoteConnector"
   Port7077" minSpareThreads="32" maxThreads="512"
              enableLookups="true" redirectPort="8443"
              acceptCount="10" debug="0" connectionTimeout="20000"
              useURIValidationHack="false"  URIEncoding="UTF-8"/> -->
```

5. Save the file.

3.2.5.3 Troubleshooting SSL-enabling the eG RUM Collector

Troubleshooting the “Certificate error” that occurs when accessing an eG manager that is SSL-enabled using a certificate from an internal CA

Typically, when you attempt to access an eG manager that has been SSL-enabled using the certificate obtained from an internal CA, the browser will throw the following error message:

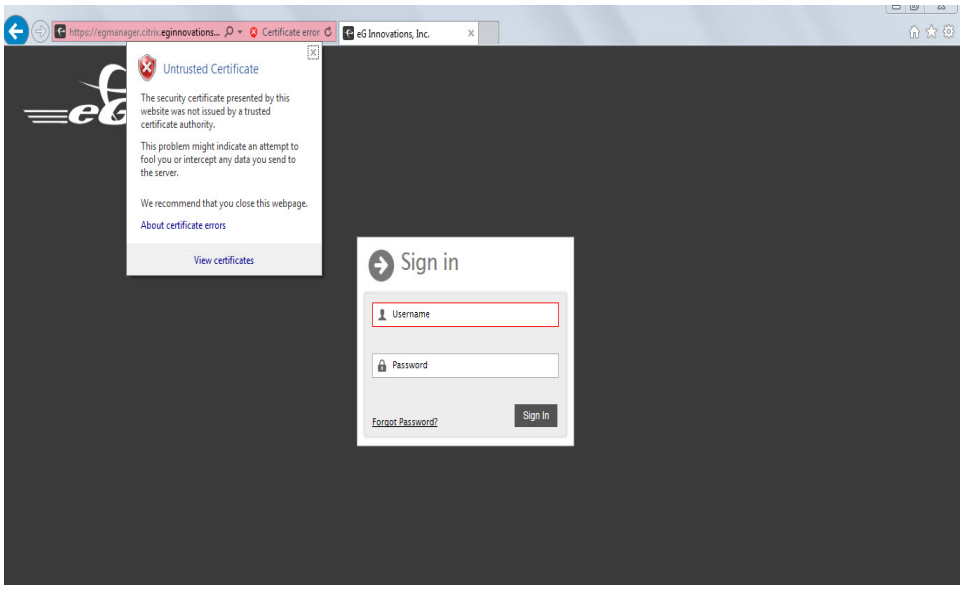


Figure 3.19: The "Certificate error" that the browser reports

To avoid this error, you will have to import the internal CA's root certificate to the browser and store it as a 'trusted root certificate'. For this, follow the broad steps outlined below:

1. Copy the internal CA's root certificate to the host from which you are accessing the eG manager (i.e., the browser host). For instance, if Microsoft Active Directory Certificate Services is your internal CA, then, you will find the root certificate of this CA on your domain server. So, in this case, you will have to copy the root certificate from the domain server to your browser host.

2. Next, using Windows Explorer, browse for the certificate, and once found, right-click on it. From the shortcut menu that appears, select the **Install Certificate** option (see Figure 3.20) to import the certificate to the browser.

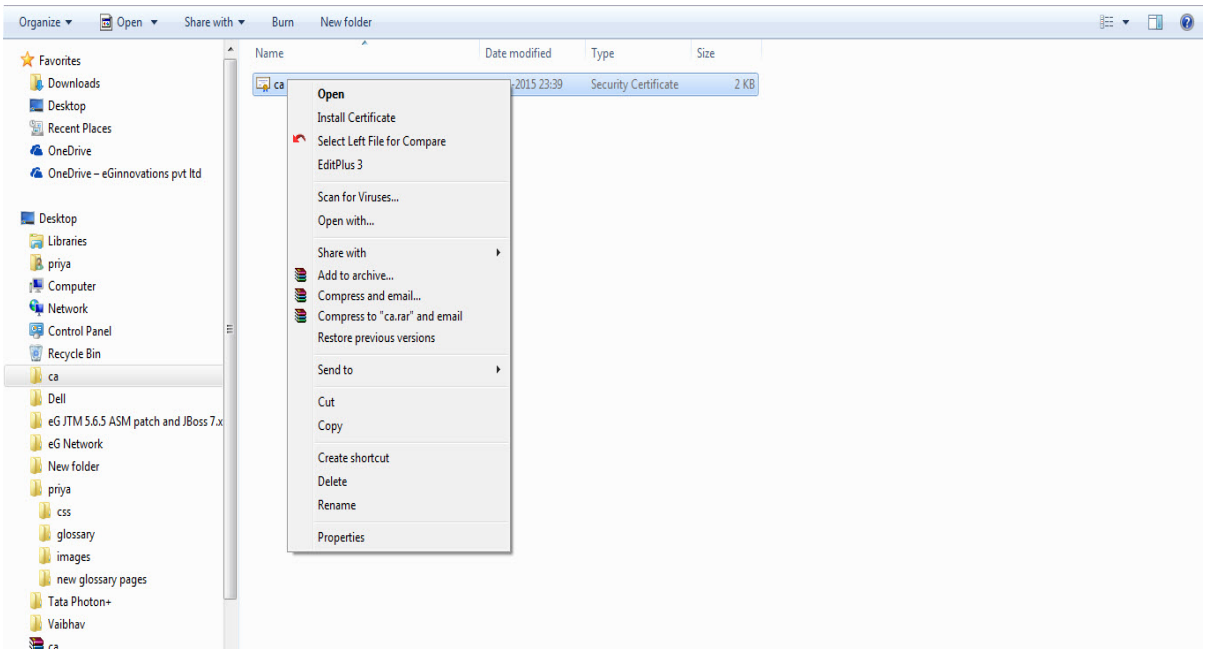


Figure 3.20: Selecting the option to install the certificate on the browser host

3. Figure 3.21 will then appear. Click **Next** here to continue.

Figure 3.21: Welcome screen of the Certificate Import Wizard

4. Figure 3.21 will then appear. Here, select the **Place all certificates in the following store** option, and click the **Browser** button to indicate where the certificate is to be stored.

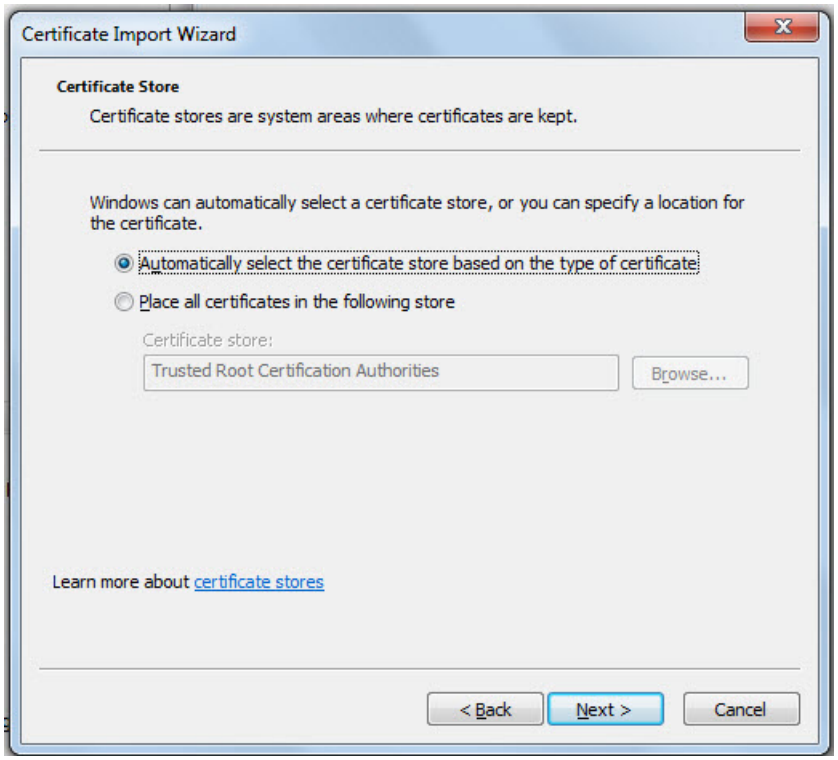


Figure 3.22: Choosing to place the certificate in a specific store

5. From Figure 3.23 that then appears, select the **Trusted Root Certificate Authorities** store and click **OK**.

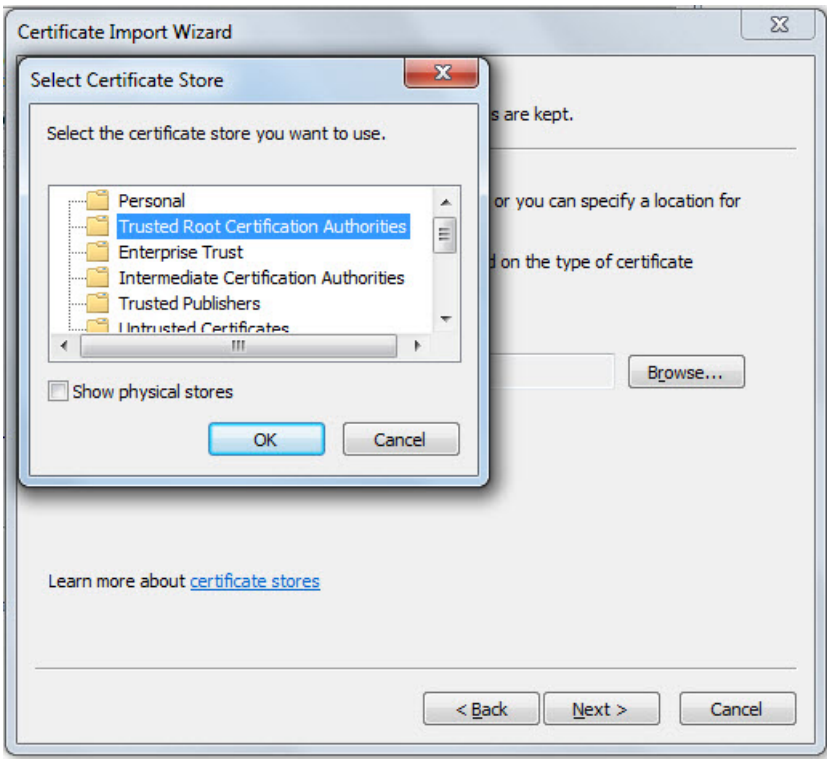


Figure 3.23: Storing the certificate in the Trusted Root Certificate Authorities store

6. The chosen store will then appear in the text box below **Place all certificates in the following store** option, as depicted by Figure 3.24. Click **Next** in Figure 3.24 to continue.

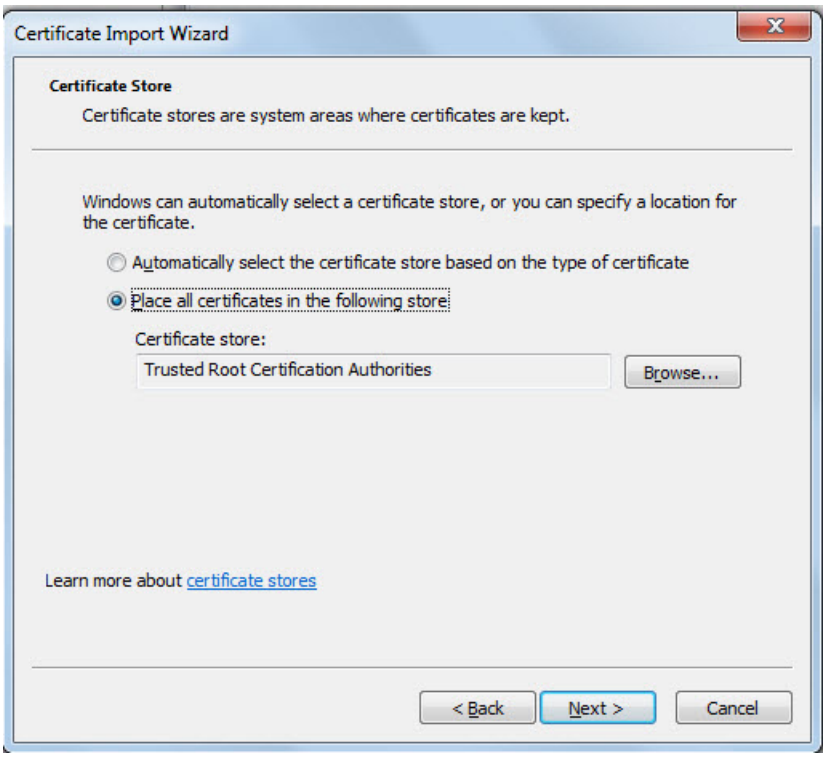


Figure 3.24: The chosen store displayed

7. A quick summary of your selections will appear in Figure 3.25. Review your specifications and click **Finish** to complete the import.

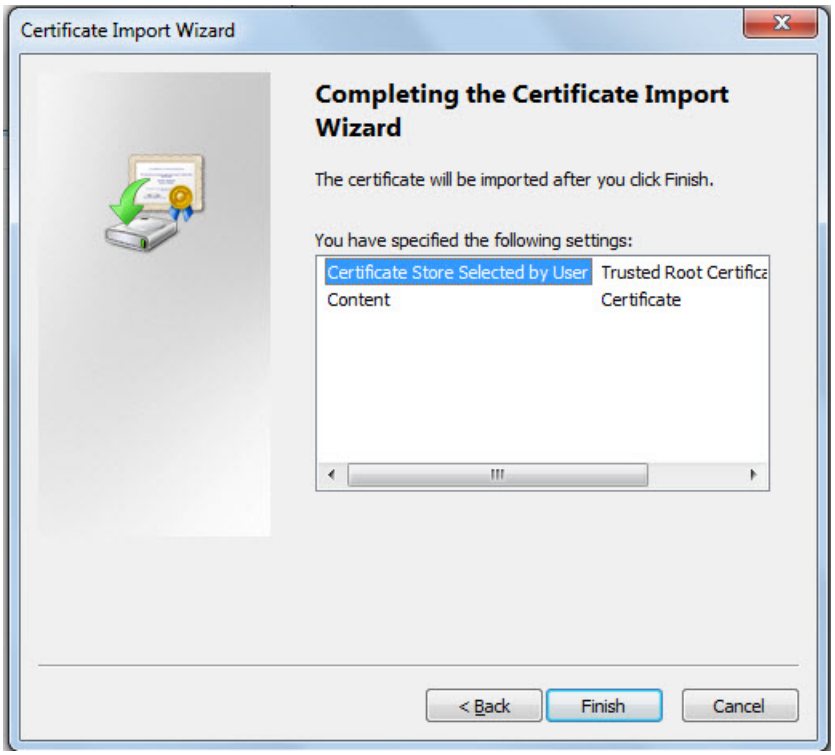


Figure 3.25: Finishing the import

8. The following warning message will appear. Click **Yes** in Figure 3.26 to proceed with the import.

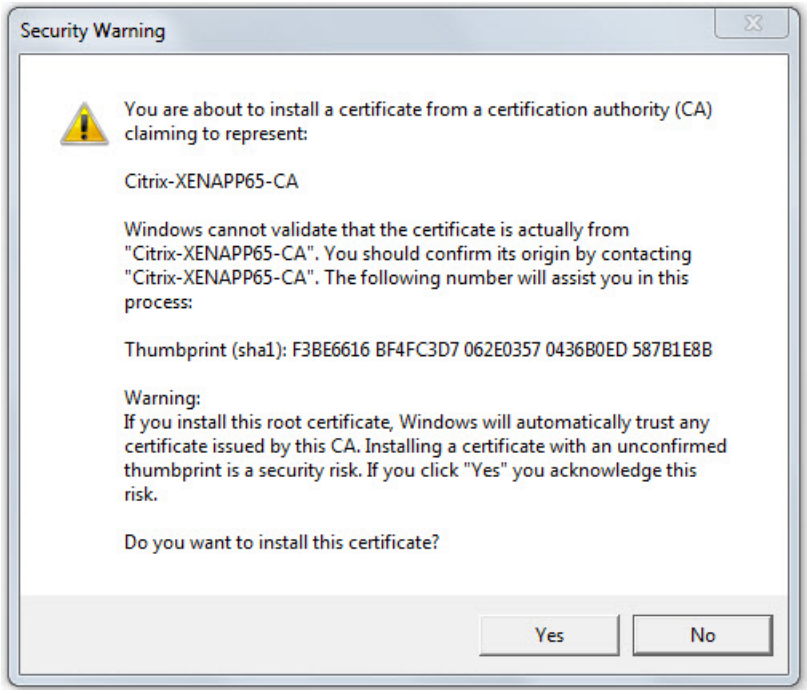


Figure 3.26: A warning message that appears when importing a certificate issued by an internal CA

9. If import is successful, the following message will appear. Click **OK.**

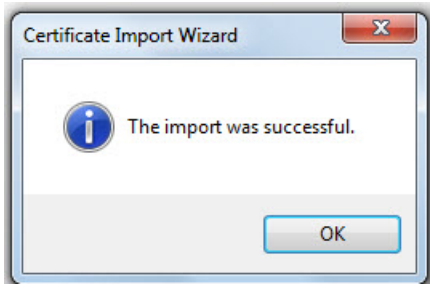


Figure 3.27: A message box informing you that the certificate has been successfully imported

You will now be able to access the eG manager without a glitch!

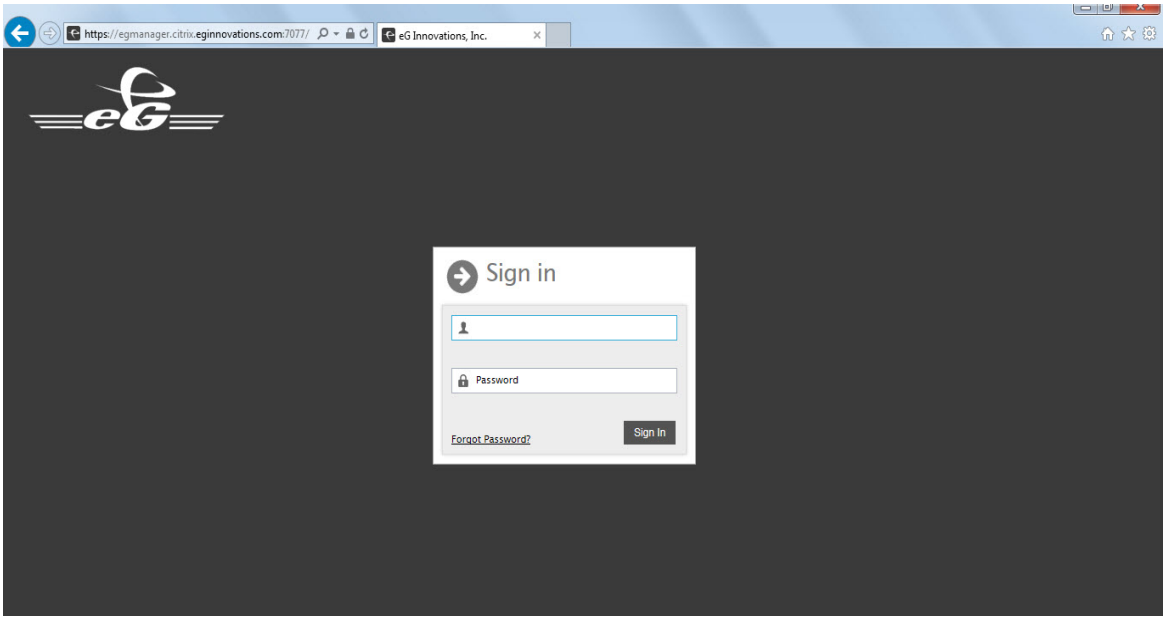


Figure 3.28: The login screen of the eG manager, without the 'Certificate error'

3.2.6 Registering the New eG RUM Collector with the eG Manager

Once the collector is installed, you need to register the collector with the eG manager. For this, follow the steps below:

1. Login to the eG administrative interface.
2. Select the **RUM collector**s options from the **Miscellaneous** tile of the **Admin** tile menu.
3. Figure 3.29 will then appear listing the collectors that pre-exist.

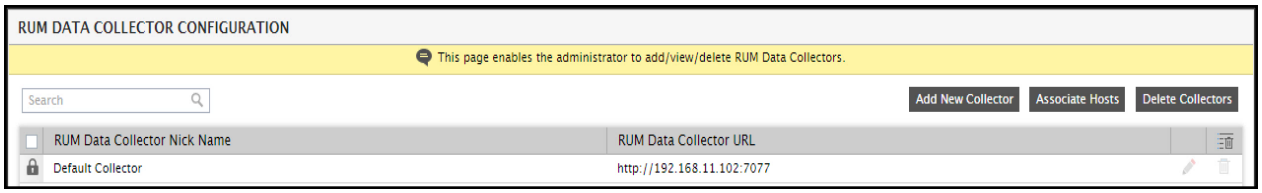


Figure 3.29: List of RUM collectors that pre-exist

4. The eG manager is bundled with a default RUM collector. By default, this default collector will be listed in Figure 3.29. **You can neither modify the details of this collector, nor delete it.**
5. To add a new collector, click the **Add New Collector** button in Figure 3.29.
6. Figure 3.30 then appears. In the **Nick Name** text box of Figure 3.30, provide a unique name for the new collector. Then, specify the **Public URL** of the RUM collector. Optionally, you can also provide a **Private URL** for the RUM collector.

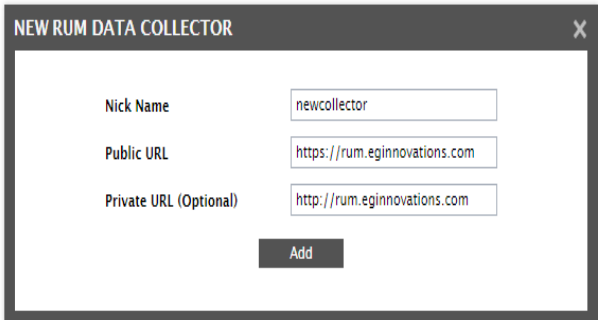


Figure 3.30: Adding a new RUM collector

Note:

A private URL is mandatory in the following situations:

- If the web site/web application being managed receives requests from users within a private network; in this case, the browsers used by these intranet users will be able to communicate user experience metrics to the RUM collector, only if the collector is configured with a private URL.
- If the remote agent monitoring the web site/web application is deployed in a private network; without a private URL, this remote agent will not be able to poll the collector for metrics.

7. Finally, click the **Add** button in Figure 3.30 to add the new collector.

3.3 Managing the Web Site/Web Application to be Monitored

Now that the required collectors have been installed and configured, let us now proceed to manage the web site/web application to be monitored and assign one of the configured collectors to it.

Note:

A collector that is **not SSL-enabled** cannot manage an SSL-enabled (i.e., an HTTPS) web site/web application. However, a collector that is **SSL-enabled**, can manage both HTTP and HTTPS web sites/web applications.

For this, follow the steps below:

1. In the eG administrative interface, select the **Add / Modify** option from the **Components** menu in the **Infrastructure** tile.

2. Figure 3.31 will then appear.

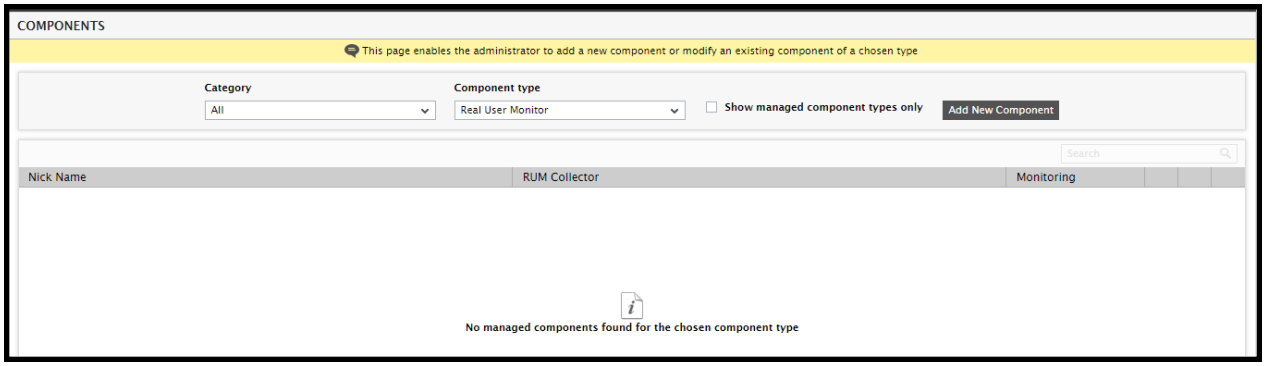


Figure 3.31: Choosing to add a Real User Monitor

3. Select **Real User Monitor** as the **Component type** from Figure 3.31 and click the **Add New Component** button to add a new component of that type.

4. When Figure 3.32 appears, provide a unique name for the web site/web application that you want to monitor in the **Nick name** text box.

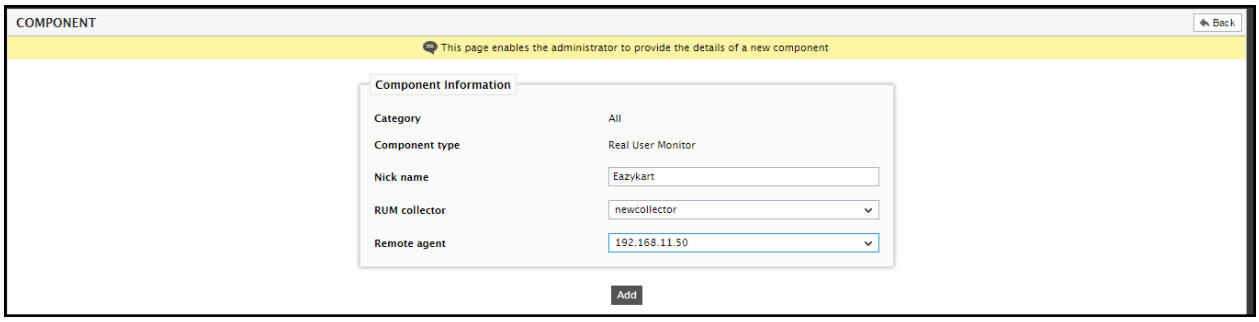


Figure 3.32: Adding a Real User Monitor

5. Then, select the RUM collector you want to assign to the specified web site/web application. By default, the **Nick Name** (see Figure 3.30) of each of the RUM collectors that you have configured in your environment will populate the **RUM collector** drop-down in Figure 3.32. From this drop-down list, select the nick name of the RUM collector that you want to assign to the web site/web application being managed.

6. Next, select the **Remote agent** that should monitor the web site/web application being managed by polling the selected collector.

7. Then, click the **Add** button in Figure 3.32 to add the *Real User Monitor* component. Doing so will invoke Figure 3.33. Figure 3.33 provides you with multiple options to instrument the web pages in the managed web site/web application, so that they can capture user experience metrics.

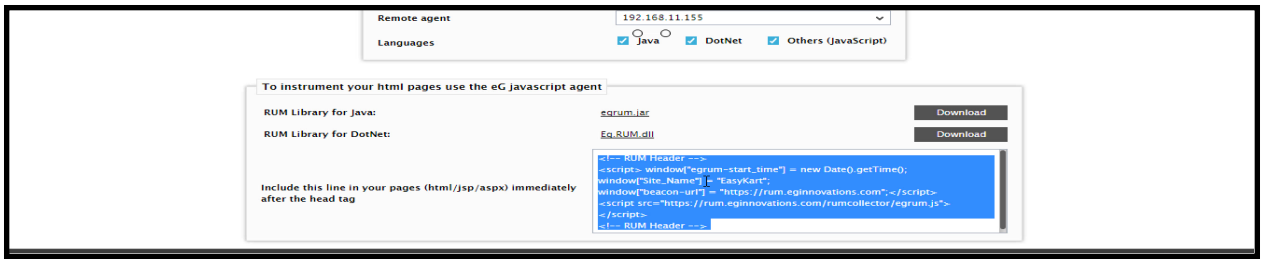


Figure 3.33: Viewing the code snippet to be injected into the monitored web pages

3.4 Instrumenting the Web Site / Web Application to be Monitored

Figure 3.34 clearly delineates the options available for RUM-enabling a web site/web application, and when each of those options become applicable.

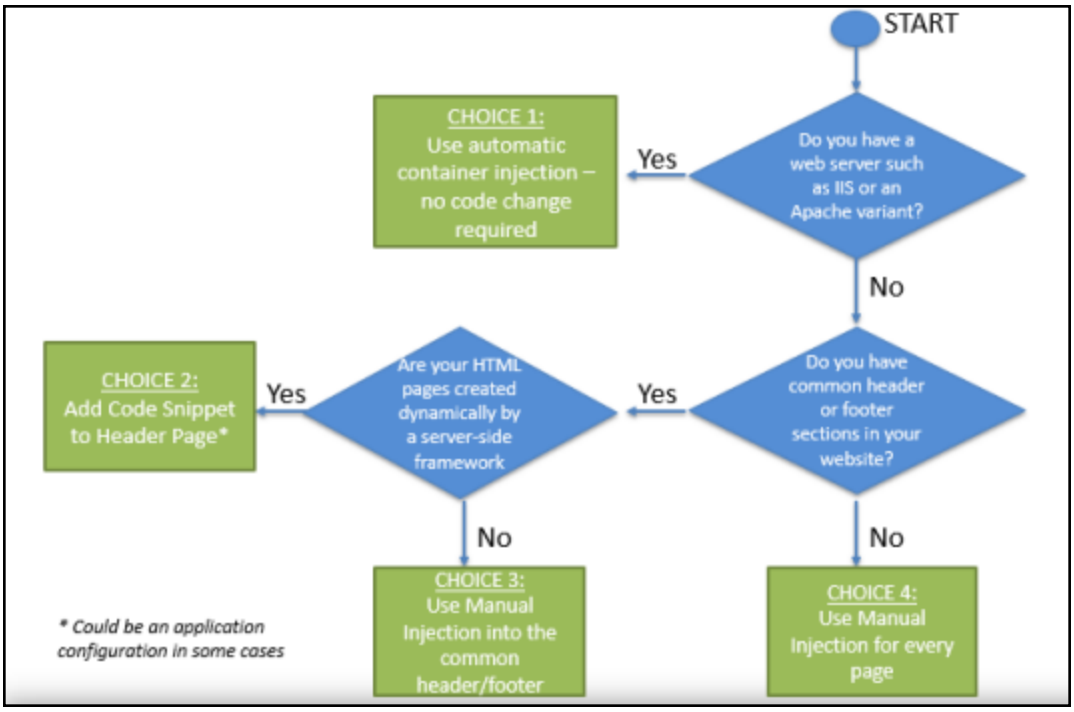


Figure 3.34: The options available for RUM-enabling a web site / web application and when those options become applicable

Each of the options depicted by Figure 1 are discussed briefly below:

- **Container assisted injection:** If the web site / web application to be RUM-enabled is hosted on an IIS / Apache web server or any of its variants (say, Oracle HTTP server), then you can use container assisted injection to RUM-enable that web site / web application. On IIS, this can be achieved by defining URL Rewrite rules. Refer to Section **3.4.1** topic for details. On Apache and Oracle HTTP server, this can

be achieved by following the steps detailed in Instrumenting an Apache Web Server Using Container Assisted Injection and Instrumenting an Oracle HTTP Server Using Container-based Injection topics.

- **Adding JavaScript code to Header page:** In the case of a web site / web application that is **not** hosted on IIS, Apache, or any variants, but has a common header/footer sections and dynamically generates HTML pages, you can add the code to the Header page of that web site / web application to RUM-enable it. This can be done in any of the following ways:
  - RUM Library for Java: **Only Java-based web sites / web applications can be instrumented using this option.** Some Java-based web sites / web applications are designed in such a way that they dynamically generate the web pages (HTML pages, JSPs, ASPX, etc.) at runtime. To 'RUM-enable' such web sites/applications, the **RUM Library for Java** can be used. This library includes a specialized **egrum.jar**, which has to be downloaded and included in the classpath of the target site/application. To download the **egrum.jar** file, click the **Download** button alongside **egrum.jar** in Figure 1.3 above. Once the download is complete, you will have to edit every page that auto-generates web pages, and add the code block that eG prescribes to each of these pages. This code block will automatically inject a JavaScript into the page where it resides. This script will track the requests to every web page that is dynamically generated by the parent page in which it has been inserted, capture the response time metrics of these requests, and send these metrics to the collector. The detailed procedure for using the**egrum.jar** is provided in Section **3.4.2** topic.

  - RUM Library for DotNet**: Only DotNet-based web sites / web applications can be instrumented using this option.** Some DotNet-based web sites / web applications are designed in such a way that they dynamically generate the web pages (HTML pages, JSPs, ASPX, etc.) at runtime. To 'RUM-enable' such web sites/applications, the **RUM Library for DotNet** can be used. This library includes a specialized **Eg_RUM.dll**, which has to be downloaded to the target web application's host, and then included in the target web application via an assembly reference. To download this library, click the **Download** button alongside **Eg_RUM.dll** in 3.4. Then, eG will provide you with a code block that you will have to insert in the **Views\Shared_Layout.cshtml** or **Default.master**file. This code block will automatically inject a JavaScript into all web pages that use the edited **cshtml** or **master** file. This script will track the requests to each of the web pages, capture the response time metrics, and send these metrics to the collector. The detailed procedure for using the **Eg_RUM.dll** is provided in Section **3.4.3**.

  - **Add JavaScript code to application configuration:** This is applicable in the case of SharePoint or PeopleSoft applications. In the case of SharePoint, you will have to insert the code in the Master page template of the application. In the case of PeopleSoft, you will have to insert the code in an Application Designer Managed Object. The steps for both the above are detailed in Section **3.4.4** and Section **3.4.5** topics.

- **Manual JavaScript injection into common header/footer:** If the web site / web application being monitored does not generate HTML pages dynamically, but supports common header/footer sections, then you can manually insert the JavaScript code into the common header/footer.
- **Manual JavaScript injection into each web page:** Any web site/web application to be monitored, regardless of the underlying development framework (be it Java, .Net, PHP, or others), can be instrumented using this option. This option is ideal in the case of web sites / web applications that **neither dynamically generate web pages, nor support a common header/footer.** In this case, you will just have to

copy the JavaScript code-snippet that eG displays in the **Include this line…** text box in 3.4 to the <*head*> tag of every web page that you want to monitor. Every time the web page is hit, the JavaScript runs to collect response time metrics and sends them to the collector. The detailed procedure for inserting the code snippet is provided in the Section **3.4.6** topic.

3.4.1 Instrumenting a Web Site or Web Application on IIS using URL Rewrite

IIS URL Rewrite 2.0 enables Web administrators to create powerful rules to implement URLs that are easier for users to remember and easier for search engines to find. By using rule templates, rewrite maps, .NET providers, and other functionality integrated into IIS Manager, Web administrators can easily set up rules to define URL rewriting behavior based on HTTP headers, HTTP response or request headers, IIS server variables, and even complex programmatic rules. In addition, Web administrators can perform redirects, send custom responses, or stop HTTP requests based on the logic expressed in the rewrite rules.

To enable the auto-injection of the JavaScript code snippet into IIS 7-based web sites/web applications, you need to use the URL Rewrite module.

Before attempting to use this module, ensure that the following pre-requisites are fulfilled:Make sure that you have access to the IIS Manager console.

- Download and install the URL Rewrite module on the IIS 7 server. To download this module, go to the URL: http://www.iis.net/downloads/microsoft/url-rewrite
- Ensure that a web site/ web application on IIS 7 has already been added as a Real User Monitor using the eG administrative interface. The JavaScript code snippet that eG provides when adding the component should be copied to a text file.
- Check whether static compression is enabled for the web site/web application that is to be RUM-enabled. To perform this check, navigate to the 'Compression' extension menu on the IIS manager. For instance, if you want to know whether/not static compression is enabled for the Default Web Site on IIS 7, click on the Default Web Site node in the tree structure in the left panel of the IIS manager (see Figure 3.35), and then select the Compression option from the right panel.

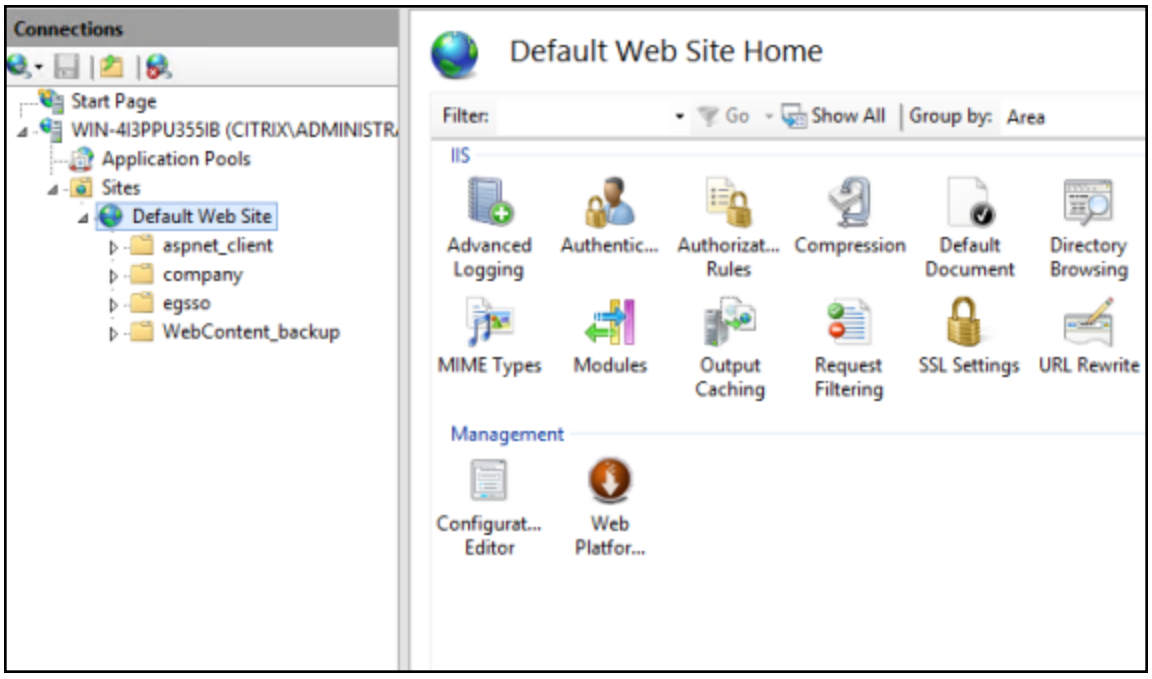


Figure 3.35: Accessing the Compression menu

- Then, when Figure 3.36 appears, check whether the Enable static content compression check box is selected; if so, it means that Static Compression has been enabled.

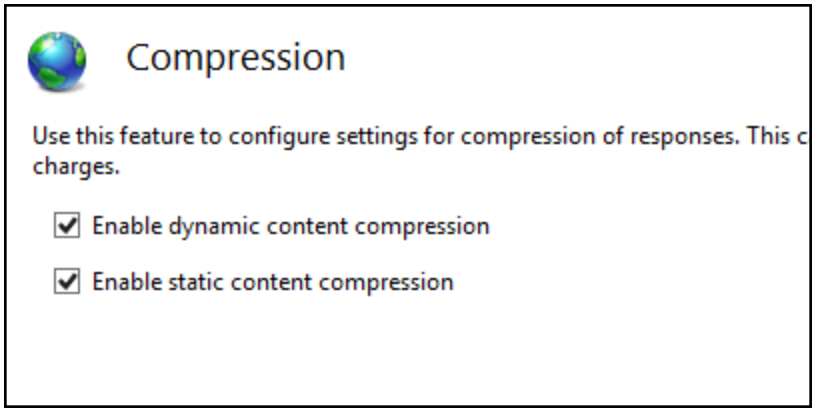


Figure 3.36: Checking whether static compression is enabled

If the Enable static content compression check box is selected in Figure 3.36, then additionally, perform the following steps:

- Open the command prompt and execute the following command:

  reg add HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\InetStp\Rewrite /v LogRewrittenUrlEnabled /t REG_DWORD /d 0

- Then, run the iisreset command.
- Next, make sure that the dynamicCompressionBeforeCache property is set to false for the

/system.webServer/urlCompression configuration element in the web.config file (see Figure 3.37).

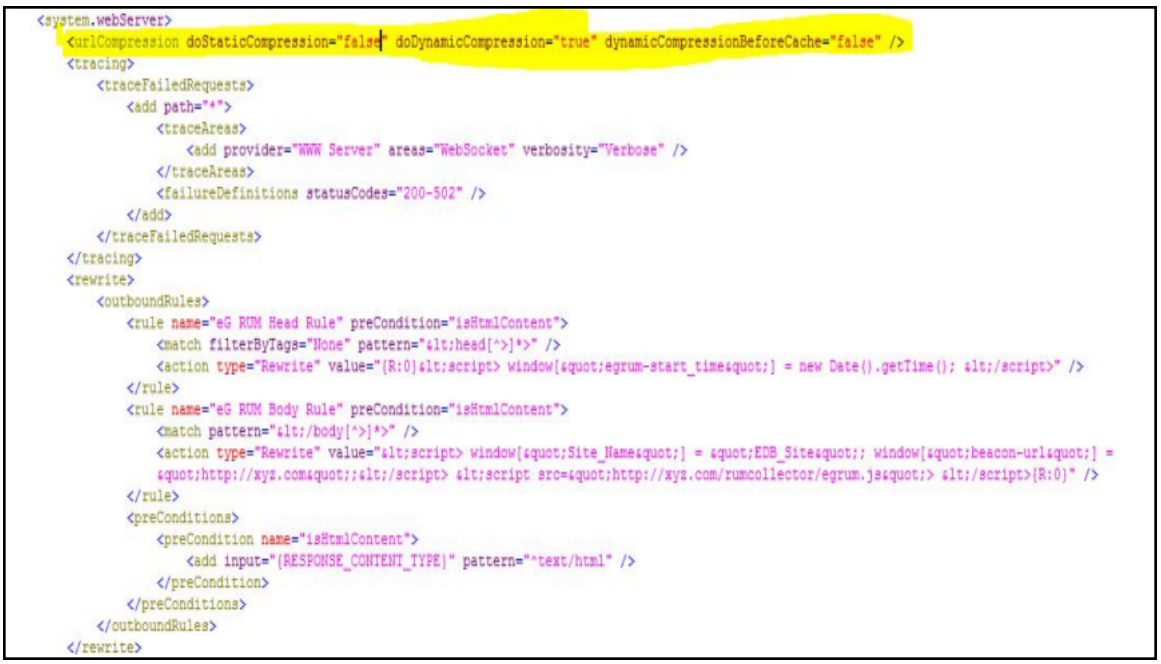


Figure 3.37: Setting the dynamicCompressionBeforeCache property to false

- Re-order the IIS modules to have URL Rewrite module (RewriteModule) run before Dynamic Compression module (DynamicCompressionModule). In other words, in the modules ordered view of the IIS Manager console, the Dynamic Compression module should be above the URL Rewrite module.

Once all the afresaid pre-requisites are fulfilled, proceed to use the URL Rewrite module. For that:

1. Create two rewrite rules on IIS 7 – a **eG RUM Head Rule** and a **eG RUM Body Rule**. Let's begin by creating the **eG RUM Head Rule**.
2. For this, first select the web site (say, Default Web Site) that is to be RUM-enabled from the tree structure in the left panel of the IIS manager console. Then, from the right panel, select the URL Rewrite option (see Figure 3.38).

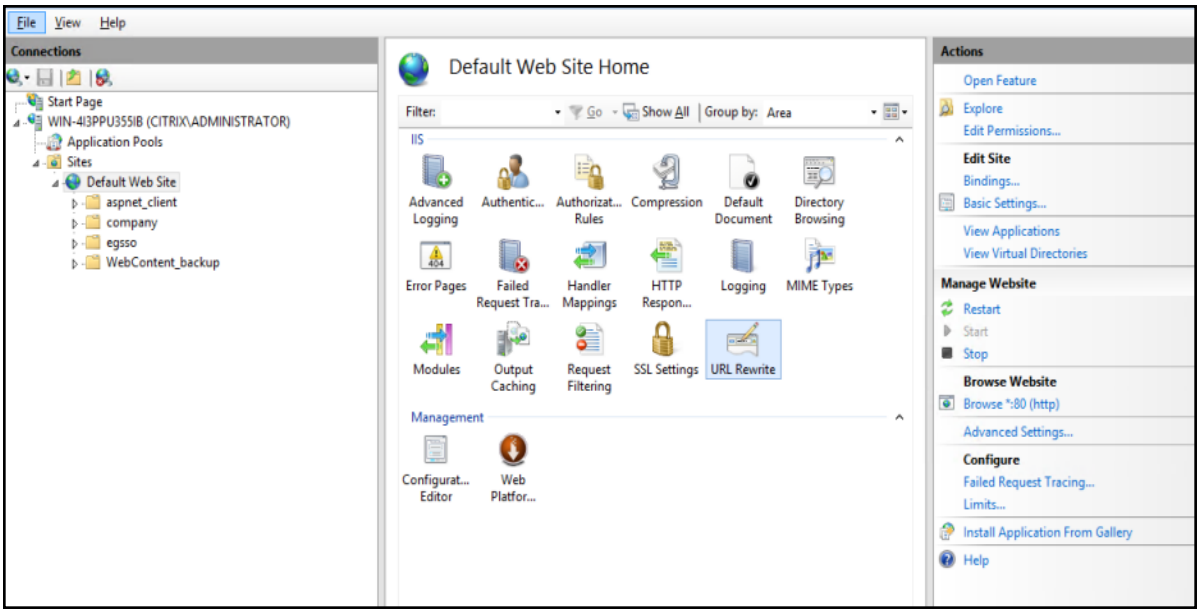


Figure 3.38: Accessing the URL Rewrite option

3. When Figure 3.39 appears, select the **Add Rule(s)** option.

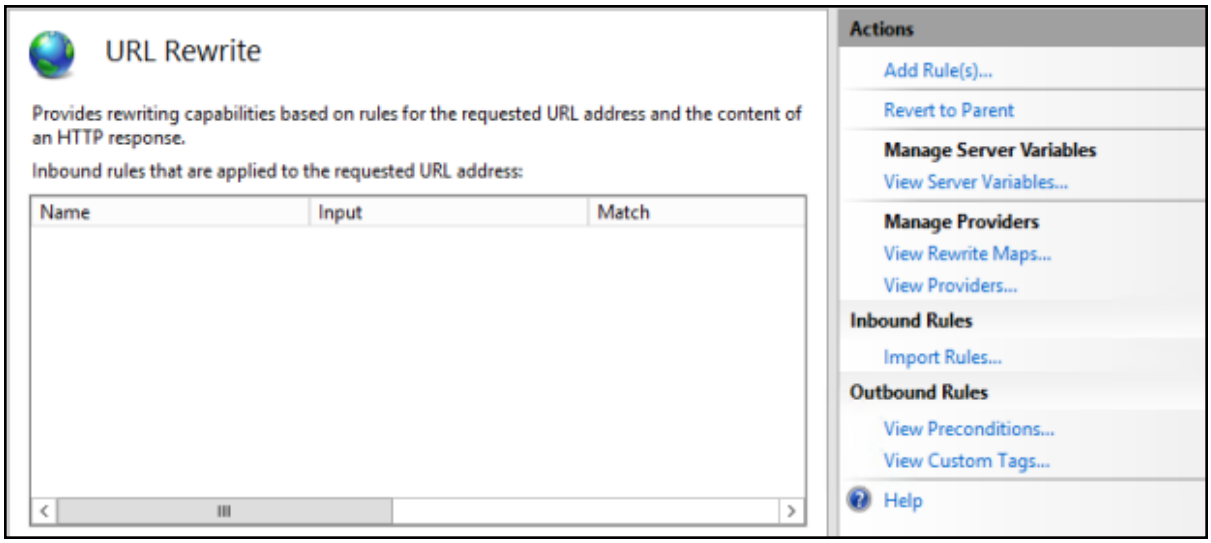


Figure 3.39: Choosing to add a new rule

4. In Figure 3.40 that then appears, click the **Blank Rule** option.

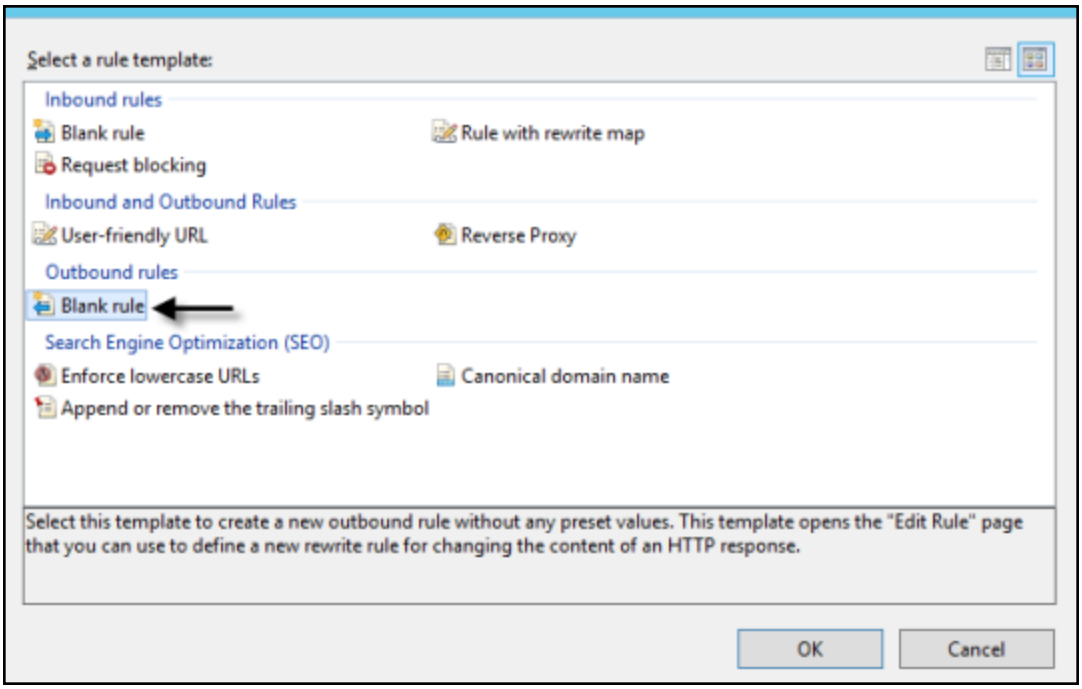


Figure 3.40: Selecting the Blank Rule option

5. Then, click the **OK** button in Figure 3.40.
6. In the dialog box that appears next, enter **eG RUM Head Rule** as the name of the new rule. From the **Precondition** list, pick the **<Create New Precondition…>** option.
7. The **Add Precondition** dialog box then appears (see Figure 3.41). Here, enter *isHtmlContent* as the **Name** of the new precondition. Then, choose **Regular Expressions** from the **Using** drop-down, and **Match All** from the **Logical grouping** drop-down. Finally, click the **Add** button.

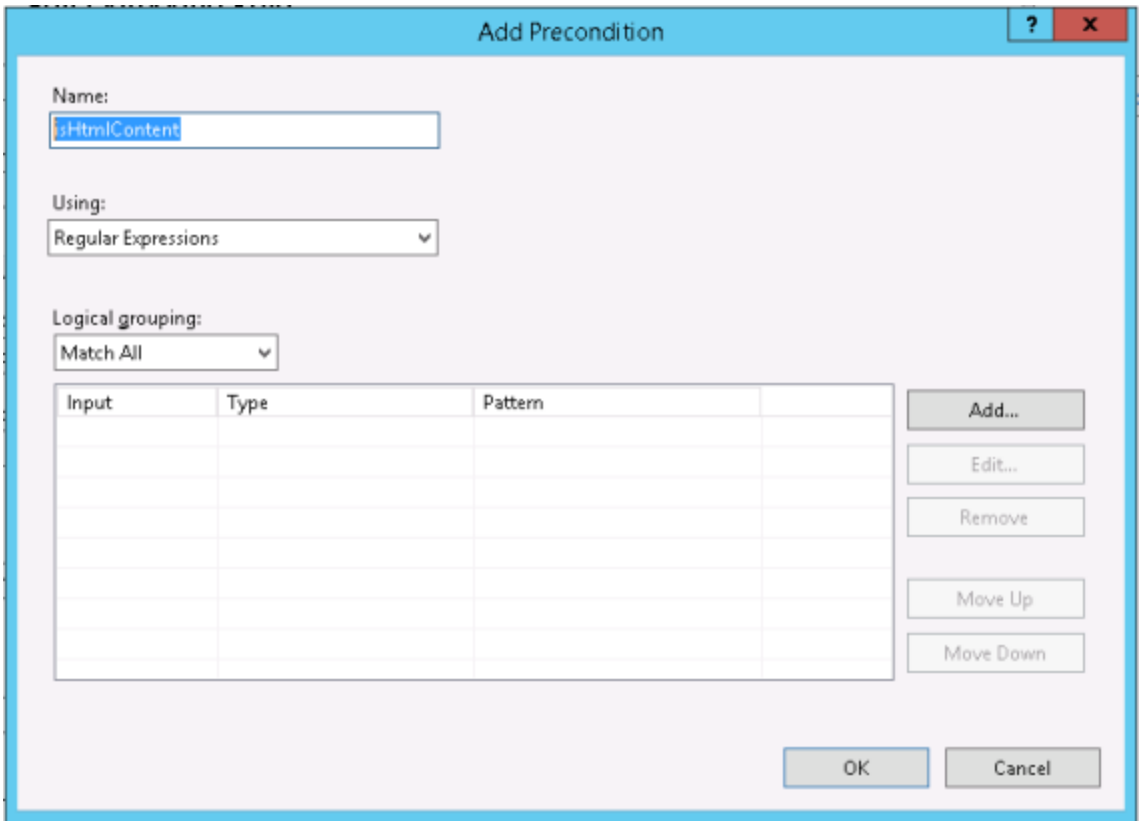


Figure 3.41: Defining the properties of the new precondition

8. When Figure 3.42 appears, type *{RESPONSE_CONTENT_TYPE}* as the **Condition** input. Then, select **Matches the Pattern** from the **Check if input string** drop-down and type *^text/html* as the **Pattern**. Finally, click the **OK** button in Figure 3.42.

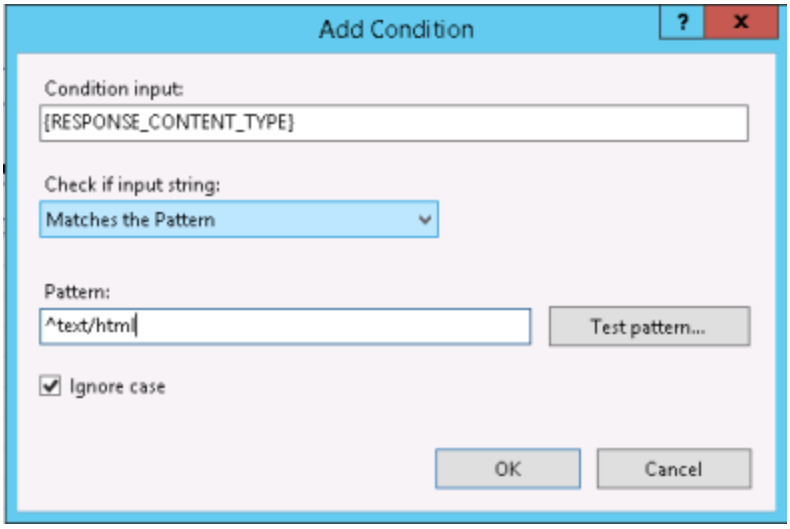


Figure 3.42: Providing the condition input, input string, and pattern

9. You will now return to Figure 3.43. Now, in the **Pattern** text box therein, type *</head[^>]*>*, as indicated by

Figure 3.43.

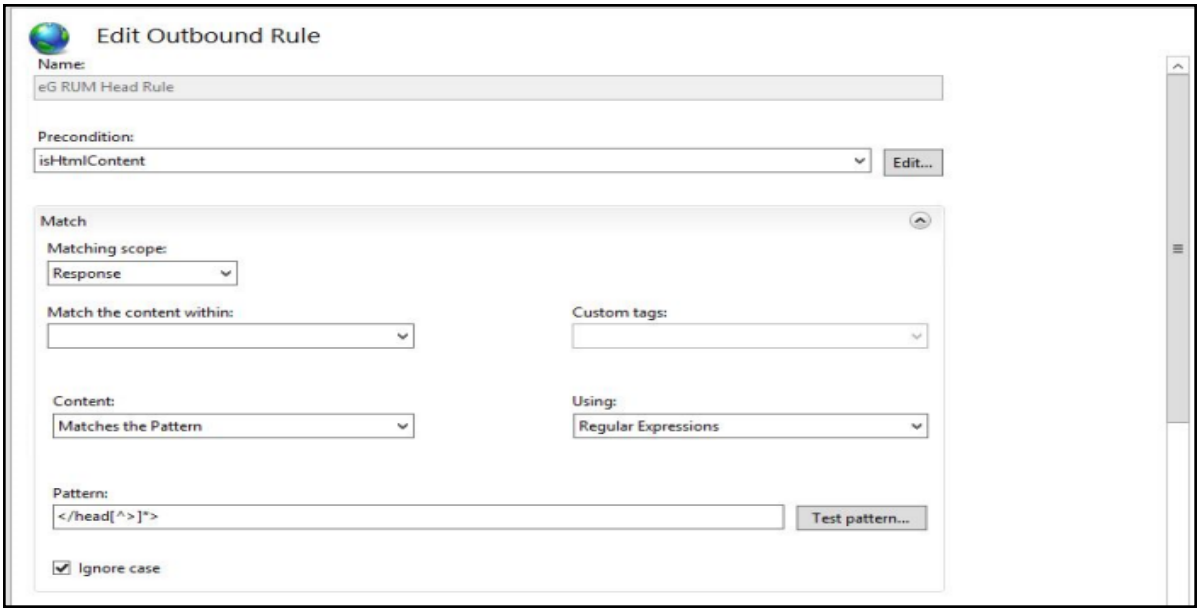


Figure 3.43: Entering the pattern for the rule

10. Then, scroll down Figure 3.44 to view the **Action Properties** section. Here, enter that portion of the eG-generated JavaScript code snippet that needs to be injected into the head (and not the body) of the web pages to be monitored. Typically, the line of code that captures the date/time at which the JavaScript code snippet began execution should be injected into the head. For instance, consider the sample JavaScript code snippet below:

```
<!-- RUM Header -->
#1:
<script> window["egrum-start_time"] = new Date().getTime(); </script>
#2 :
<script> window["Site_Name"] = "EDB_Site";
window["beacon-url"] = "http://xyz.com";</script>
<script src="http://xyz.com/rumcollector/egrum.js"> </script>
<!-- RUM Header -->
```

Here, the line of code highlighted in Bold is the one that needs to be auto-injected into the head of all web pages of the chosen web site.

To ensure this, type this entire line of code in the **Action Properties** section of Figure 3.44. Make sure that you append the line with the entry {R:0}. So, your complete Action Properties specification will be as follows:

<script> window["egrum-start_time"] = new Date().getTime(); </script>{R:0}

Figure 3.44: Specifying the line of code to be auto-injected in the head

11. Finally, click the **Apply** button in Figure 3.45.

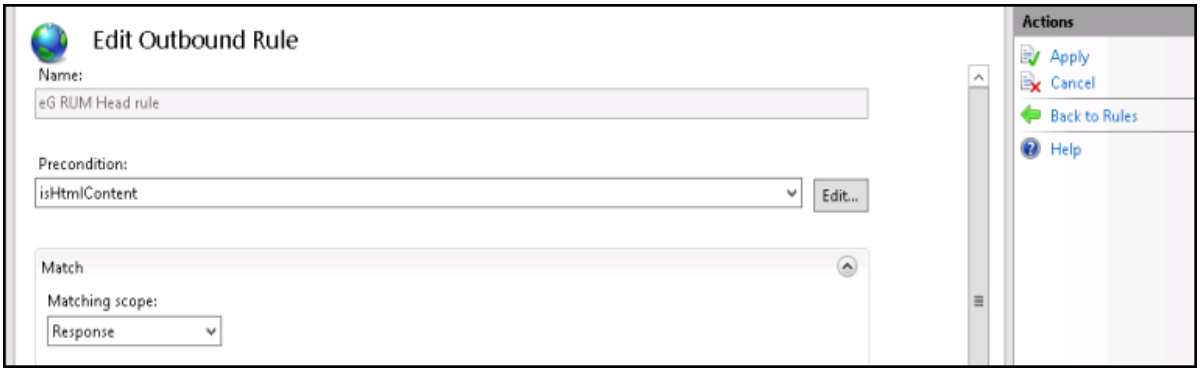


Figure 3.45: Applying the changes

12. With that the **eG RUM Head rule** has been defined. Now, let us proceed to define the next rule – the eG RUM Body rule. For that, once again select the web site (say, Default Web Site) that is to be RUM-enabled from the tree structure in the left panel of the IIS manager console. Then, from the right panel, select the **URL Rewrite option** (see Figure 3.46).

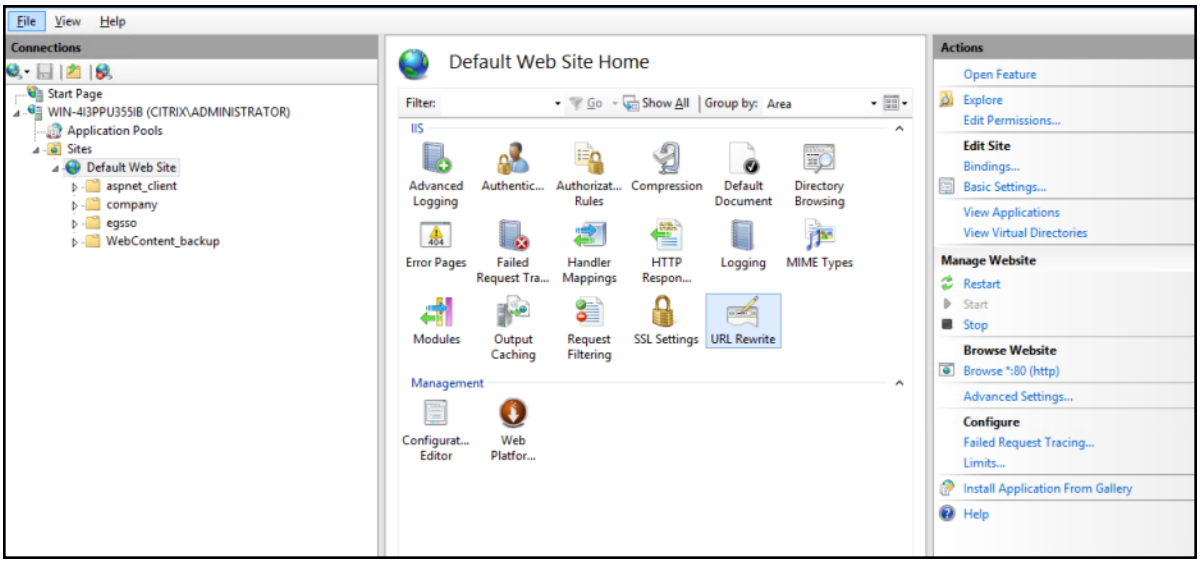


Figure 3.46: Accessing the URL rewrite option

13. When Figure 3.47 appears, select the **Add Rule(s)** option.

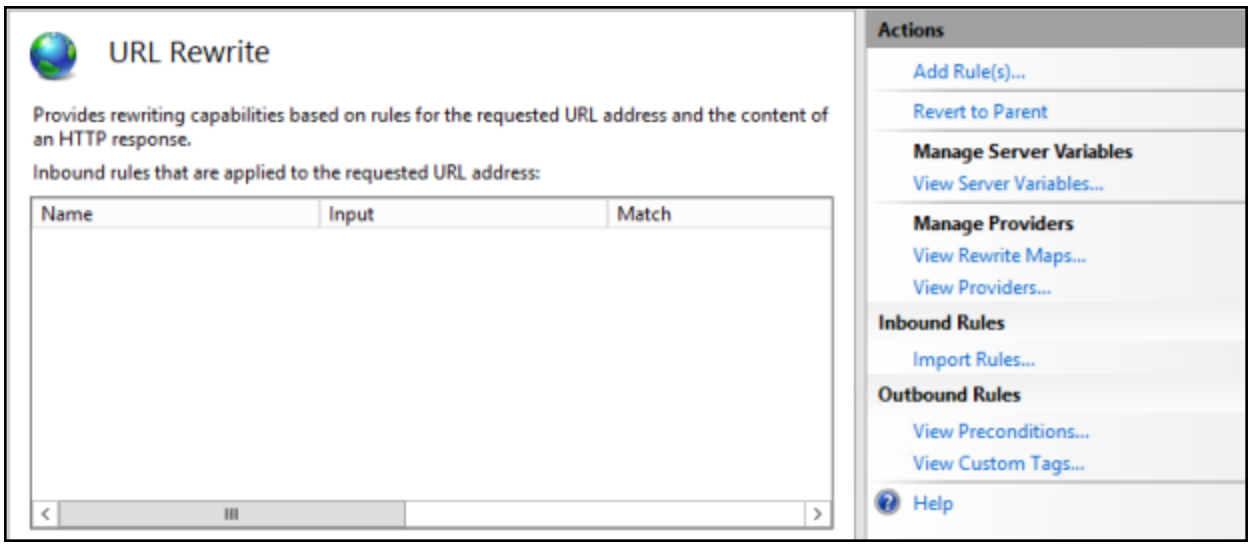


Figure 3.47: Choosing to add a new rule

14. In Figure 3.48 that then appears, click the **Blank Rule** option.

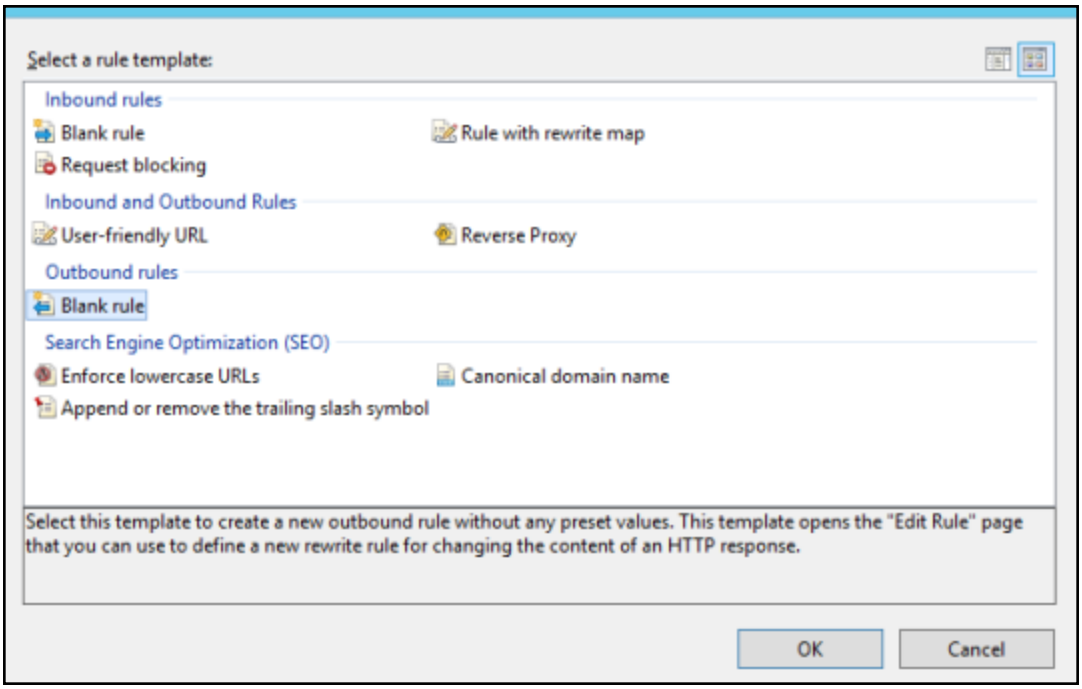


Figure 3.48: Selecting the Blank Rule option

15. Then, click the **OK** button in Figure 3.48.
16. In Figure 3.49 that appears next, enter **eG RUM Body Rule** as the name of the new rule. From the **Precondition** list, pick the **isHtmlContent** option. Then, against Pattern specify *</body[^>]*>*.

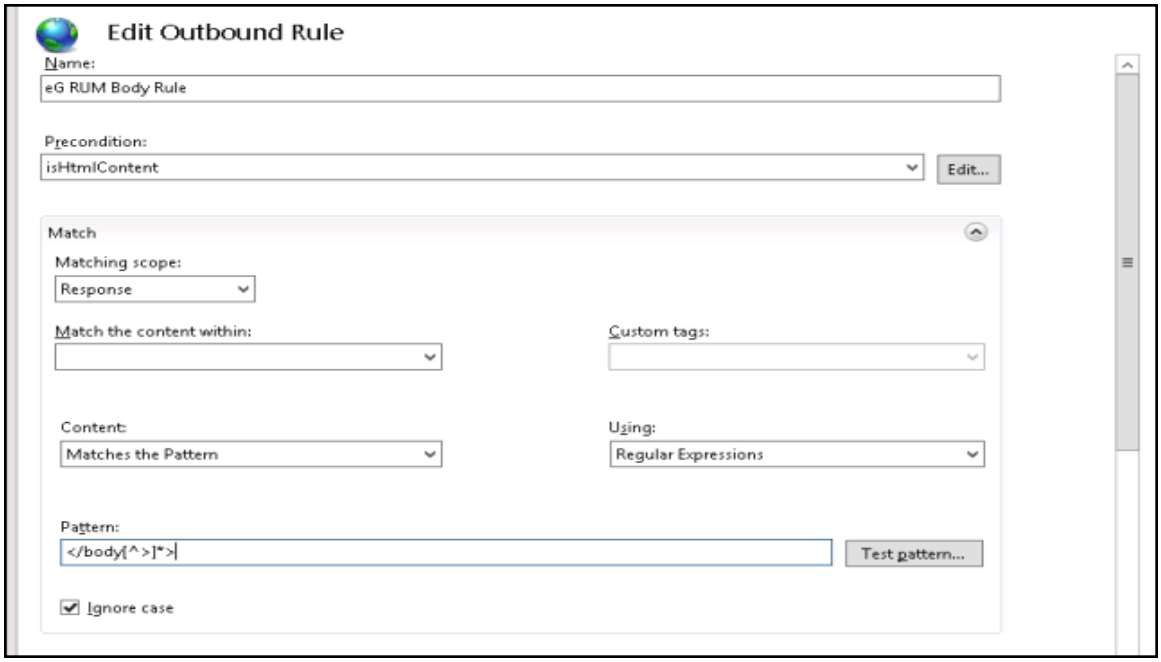


Figure 3.49: Adding the eG RUM Body Rule

17. Then, scroll down Figure 3.49 to view the **Action Properties** section. Here, enter that portion of the eG-

generated JavaScript code snippet that needs to be injected into the body (and not the head) of the web pages to be monitored. Typically, the line of code that connects to the eG RUM Collector and runs the egru,.js script should be injected into the body. For instance, consider the sample JavaScript code snippet below:

```
<!-- RUM Header -->
#1:
<script> window["egrum-start_time"] = new Date().getTime(); </script>
#2 :
<script> window["Site_Name"] = "EDB_Site";
window["beacon-url"] = "http://xyz.com";</script>
<script src="http://xyz.com/rumcollector/egrum.js"> </script>
<!-- RUM Header -->
```

Here, the line of code highlighted in **Bold** is the one that needs to be auto-injected into the body of all web pages of the chosen web site.

To ensure this, type this entire line of code in the **Action Properties** section of Figure 3.50. Make sure that you append the line with the entry *{R:0}*. So, your complete **Action Properties** specification will be as follows:

<script> window["Site_Name"] = "EDB_Site";

window["beacon-url"] = "http://xyz.com";</script>

<script src="http://xyz.com/rumcollector/egrum.js"> </script>{R:0}

Figure 3.50: Specifying the line of code to be auto-injected in the body

18. Finally, click the **Apply** button in Figure 3.51.

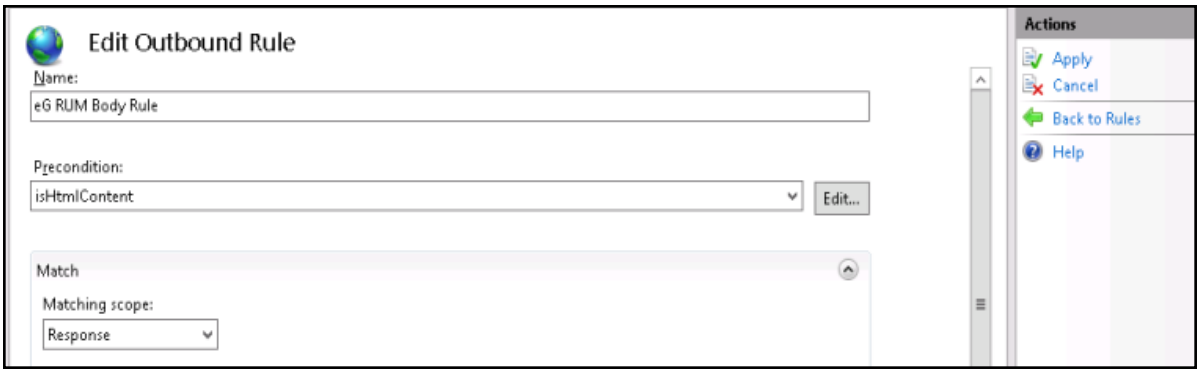


Figure 3.51: Applying the changes

19. Once both the rules have been created, click **Browse**. After the page loads successfully, view the source code of that web page to ensure that the JavaScript code snippet has been injected as expected. Figure 3.52 depicts a sample web page, where both the URL Rewrite rules have injected the correct line of code in the correct places.

Figure 3.52: Checking whether the rules have injected the code correctly

Alternatively, you can also auto-inject the JavaScript code snippet into only those web pages that match specific URL patterns. To achieve this, in both the rules, add conditions to match the URL patterns, as depicted by Figure 3.53.

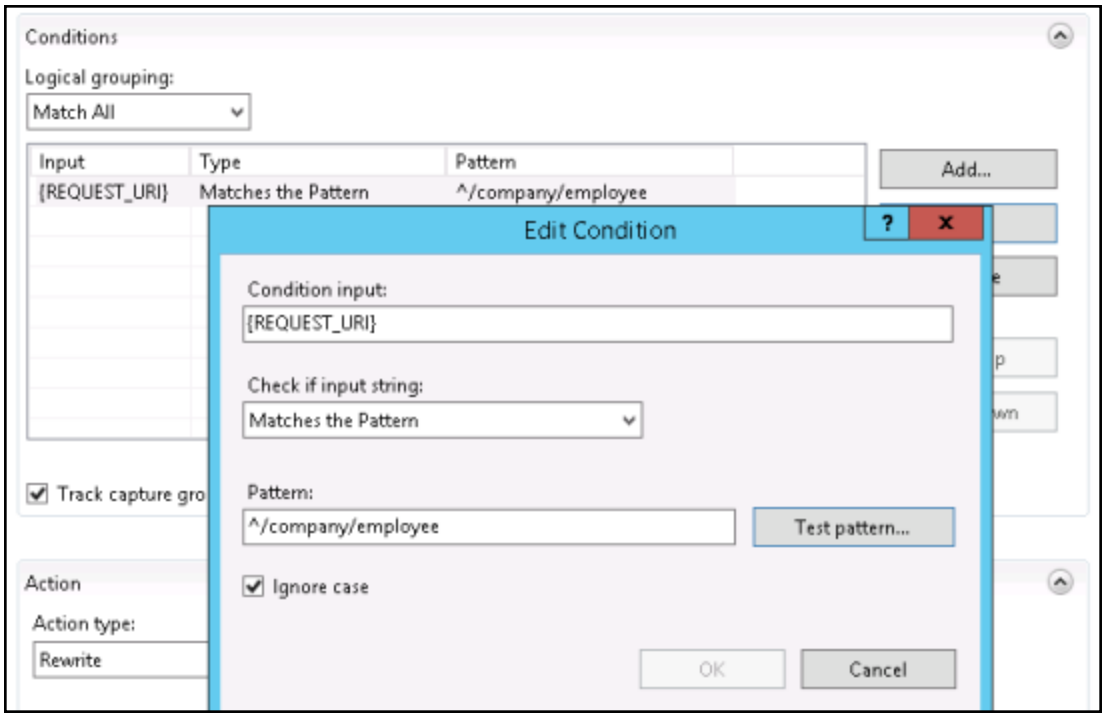


Figure 3.53: Auto-injecting the code snippet in web pages that match specific URL patterns

3.4.2 Instrumenting a Java-based Web Site / Web Application

As already mentioned, you can use the **egrum.jar** file that eG provides to instrument a Java-based web site/web application that dynamically generates web pages. To perform this instrumentation, do the following:

1. First, download the **egrum.jar** to the system hosting the web application. For this, click the **Download** button alongside **egrum.jar** in Figure 3.33.

2. Once the download is complete, make sure that the **egrum.jar** is included in the class path of the target web application. If you are using an IDE, copy it into your build project and make sure it's accessible in your runtime classpath.

3. Next, open the page that dynamically generates web pages, in an editor. In the header of the page, insert the code that imports the helper class – *com.eg.RUMHandler* – into the page, as shown in **Red** below:

```
<%@ page language="java" contentType="text/html" import="com.eg.RUMHandler;" %>
```

```
<html>
```

4. This helper class will automatically inject the JavaScript code snippet into the page.

5. To expose the output of the *RUMHandler* method, insert the line of code shown below, just after any *<meta>* tags in your *<head>* block. **Note that this code should be the first non-meta tag inside the <head> block**.

```
<%@ page language="java" contentType="text/html" import="com.eg.RUMHandler;" %>
```

```
<html>
<head>
<meta ... >
<!-- RUMHandler method return strings so you need to expose their output as shown -->
<% out.println(RUMHandler.getHeader()); %>
...
</head>
<body>
...
</body>
</html>
```

6. Save the edited page and restart the web site/web application being monitored.

7. Now, test your web application locally and view the page source in your browser. The page markup should include **< !-- RUM Header -->.**

8. Repeat steps 3 to 7 above for each page in your web site/web application that dynamically generates web pages.

Once this is done, then the auto-injected JavaScript will track requests to each of the dynamically generated web pages and report response time metrics it collects to the RUM collector.

3.4.3 Instrumenting a DotNet-based Web Site / Web Application

As already mentioned, you can use the **Eg_RUM.dll** file that eG provides to instrument a DotNet-based web site/web application that dynamically generates web pages. To perform this instrumentation, do the following:

1. First, download the **Eg_RUM.dll** to the system hosting the web application. For this, click the **Download** button alongside **Eg_RUM.dll** in Figure 3.33.
2. Once the download is complete, make sure that the **Eg_RUM.dll** is included in the target web application, and add an assembly reference.
3. Next, in an editor, open the **Views\Shared_Layout.cshtml** file for ASP .NET MVC or the **Default.master** file for ASP .NET WebForms.

   Once the download is complete, make sure that the **Eg_RUM.dll** is included in the target web application, and add an assembly reference.

   Next, in an editor, open the **Views\Shared_Layout.cshtml** file for ASP .NET MVC or the **Default.master** file for ASP .NET WebForms.

   To the **Views\Shared_Layout.cshtml** file, copy the following line of code:

```
@Html.Raw(Eg.RUM.Header()) <!--HtmlHelper.Raw prevents RUM markup from being html-encoded.-->;
```

To the **Default.master** file, copy the following line of code:

```
<%= Eg.RUM.Header() %>
```

4. Save the respective files.

5. Restart the target web application.

6. Test your web application locally and view the page source in your browser. The page markup should include **< !-- RUM Header -->.**

Once this is done, then the auto-injected JavaScript will track requests to each of the web pages that use the **Shared_Layouts.html** or **Default.master** files (as the case may be).

3.4.4 Instrumenting SharePoint

To inject the JavaScript into the pages of a Sharepoint site, do the following:

1. First, add the Sharepoint site as a *Real User Monitor* component in eG.

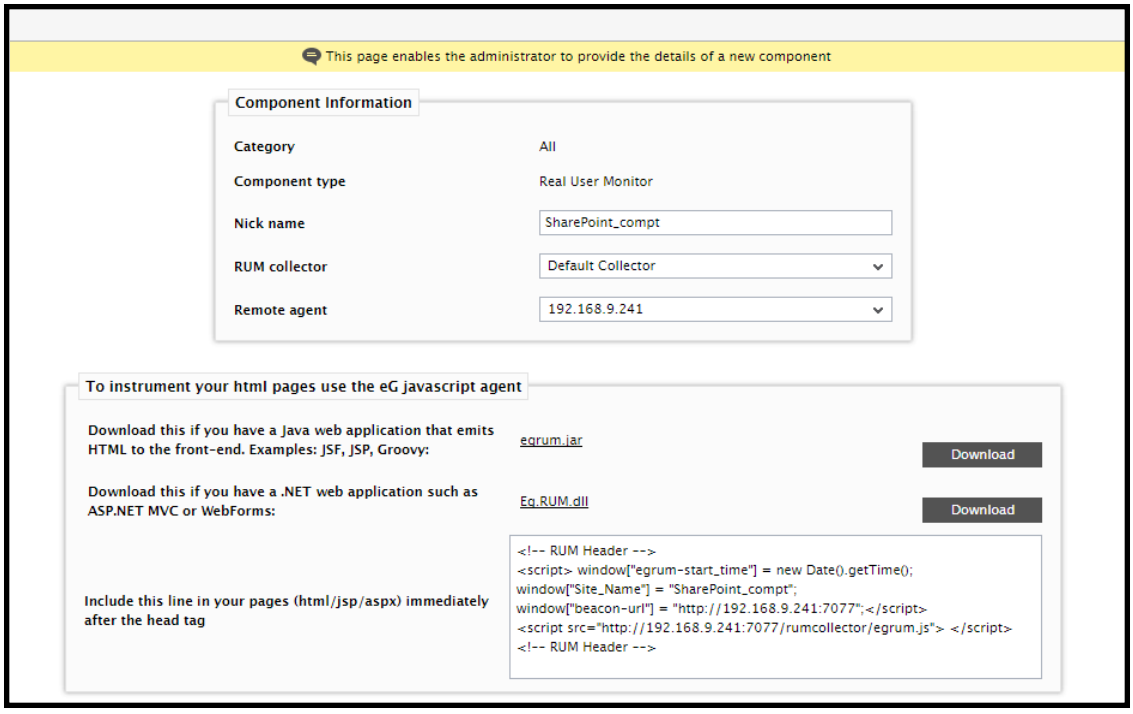


Figure 3.54: Adding the Sharepoint site as a Real User Monitor component

2. When adding the *Real User Monitor* component, you will be provided with the Javascript (as indicated by Figure 3.54 above) that you need to inject into the pages of your site.

3. Copy this script to Notepad or any other Editor.

4. Next, login to the Sharepoint site that you want to monitor as *administrator*.

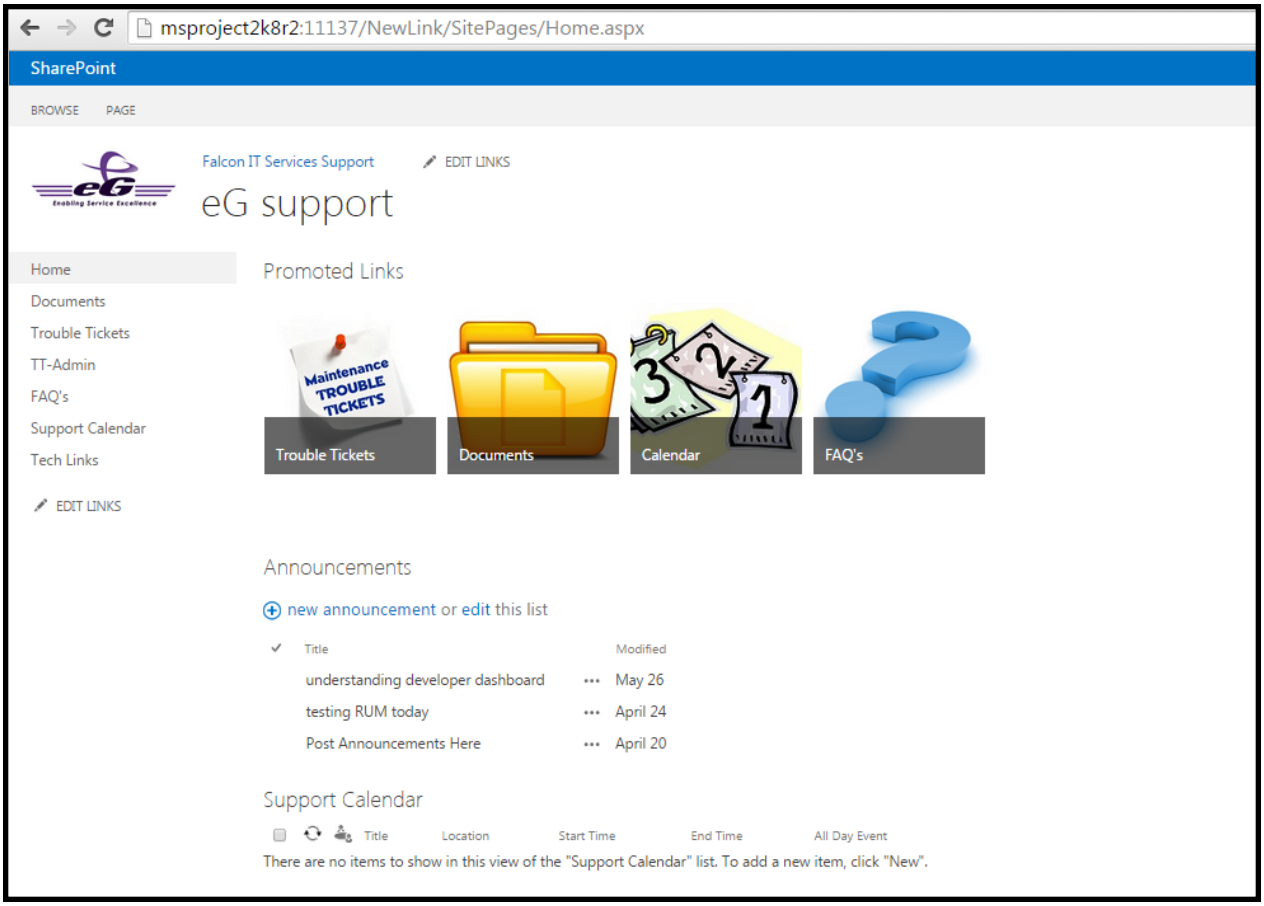


Figure 3.55: The Home page of the Sharepoint site to be monitored

5. Then, click on the **Settings** icon (indicated by Figure 3.56) at the right, top corner of the Home page to invoke a menu. From this menu, select the **Site settings** option.

Figure 3.56: Selecting the Site settings option from the Home page

6. From the **Site Settings** window (see Figure 3.57) that then appears, select the **Master Pages** option in the **Web Designer Galleries** section.

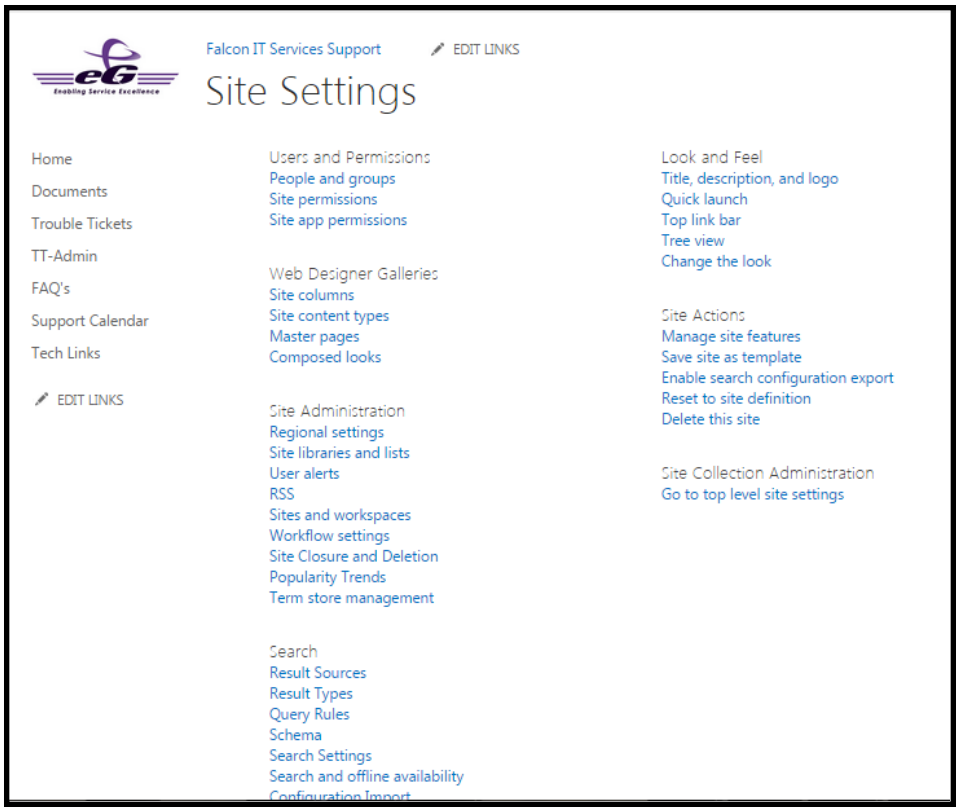


Figure 3.57: Selecting the Master pages option

7. When the **Master Page Gallery** opens (see Figure 3.58), look for the **seattle.master** file (in the case of Sharepoint 2013) or the **v4.master** file (in the case of Sharepoint 2010), in the gallery. Once you find it, hover your mouse over the file name in the gallery. A down-arrow icon (as indicated by Figure 3.58), will then become visible alongside. Click on the down-arrow icon.

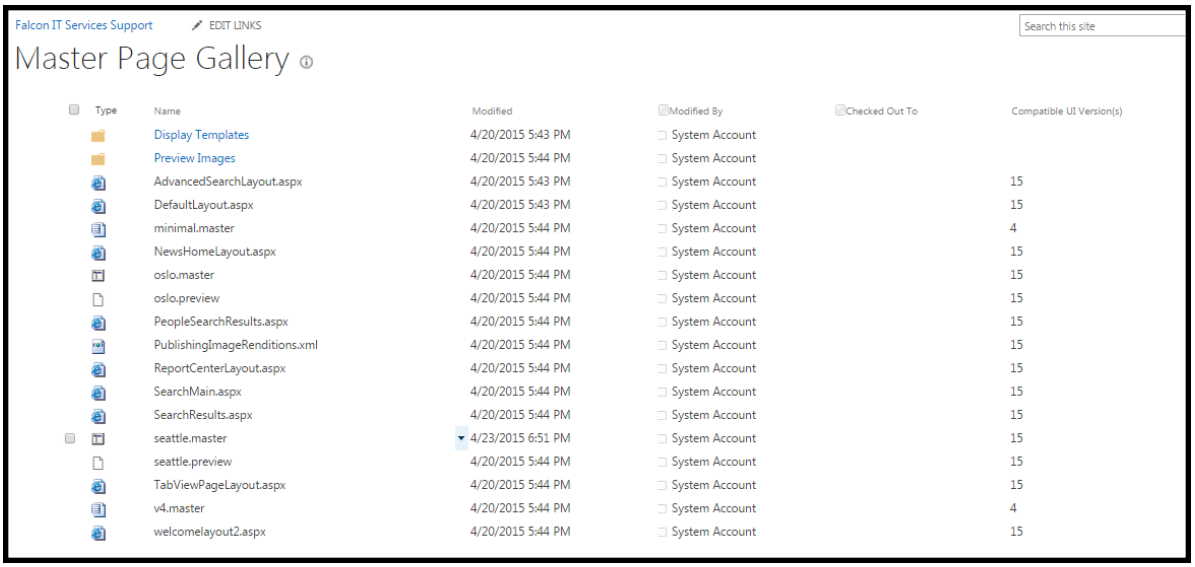


Figure 3.58: Locating the seattle.master file on Sharepoint 2013

8. Clicking on the down-arrow will invoke a menu. From this menu, select the **Check Out** option (see Figure 3.59).

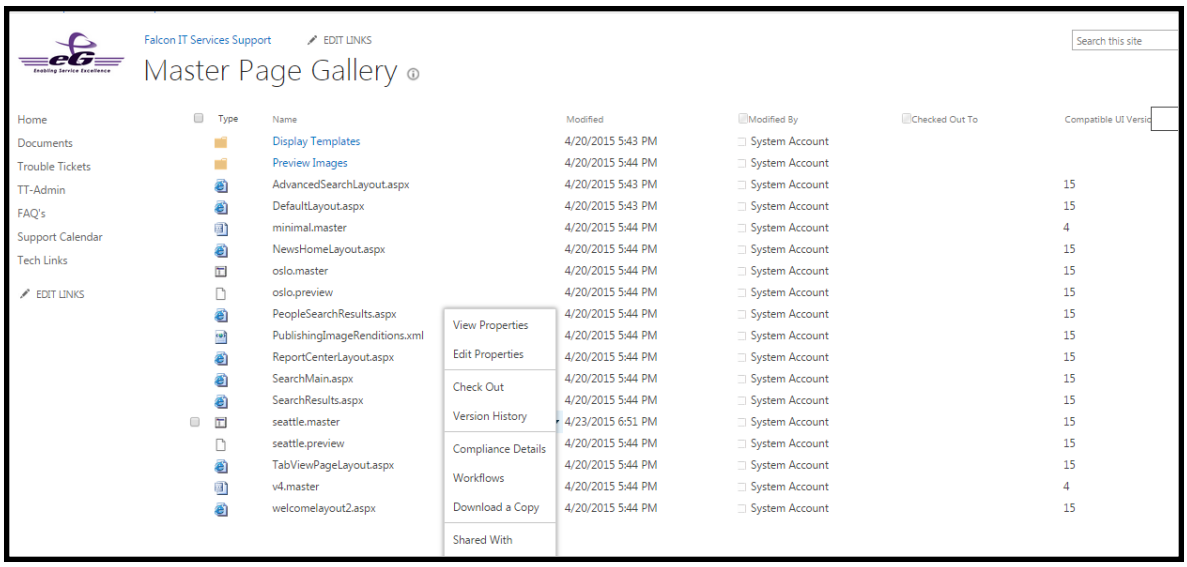


Figure 3.59: Checking out the master file

9. Then, select the **Download a Copy** option from the same menu (see Figure 3.60) to download a copy of the master file to any location on the local host.

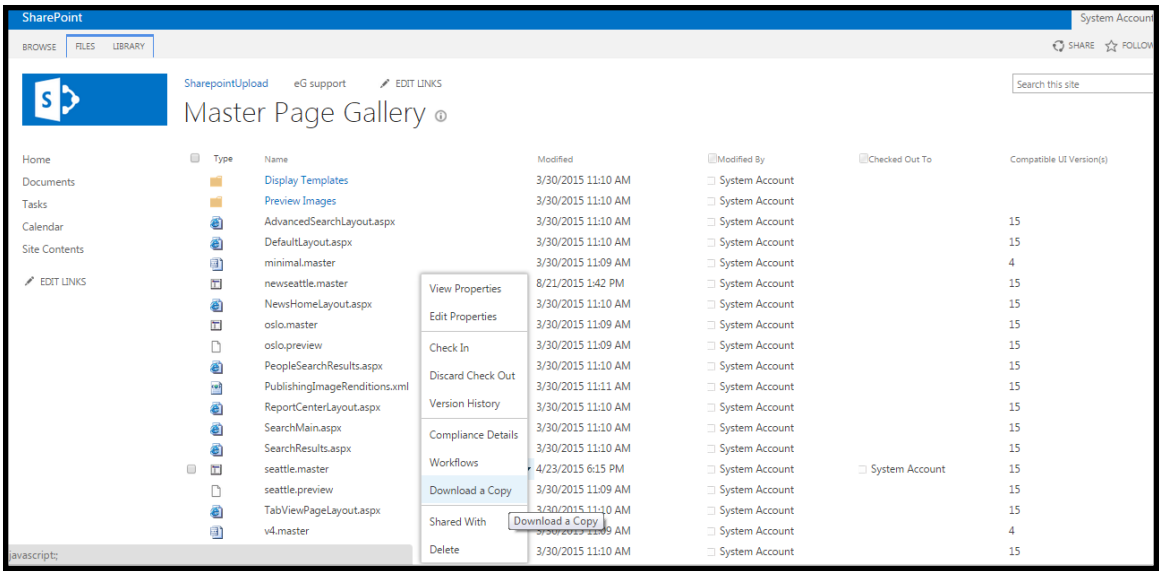


Figure 3.60: Downloading a copy of the master file

10. Next, open the downloaded copy of the master file in an Editor.

11. Paste the Javascript code that you had copied to an Editor at step c of this procedure just above the **</head>** tag of the master file (as indicated by Figure 3.61).

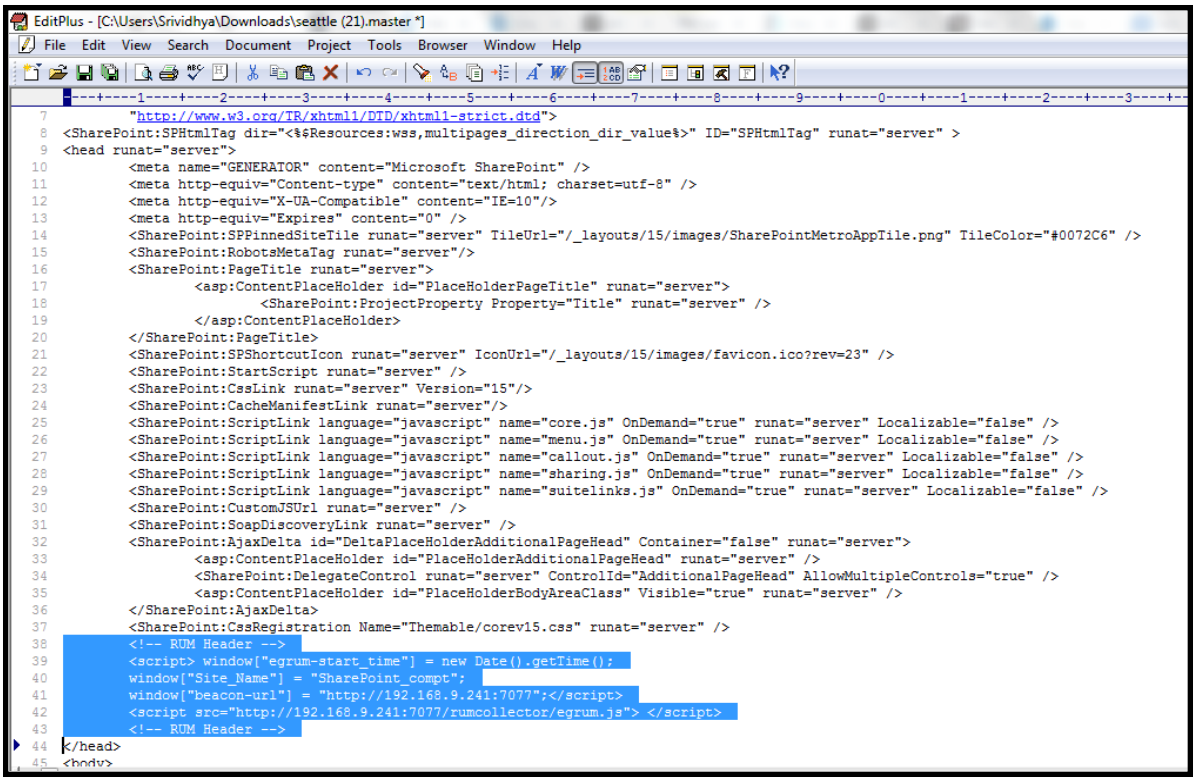


Figure 3.61: Inserting the Javascript code in the master file

Note:

Make sure that the Javascript code snippet is copied to the master file with utmost diligence. **Do not change any other content in the master file when copying the script, as it may lead to undesirable results.**

12. Save the file.

13. Then, return to the **Master Page Gallery** of Figure 3.60. From the menu on top of the gallery, select the **Files** option. Pick the **New** menu from the **Files** menu, and then choose the **Upload Document** option from it to upload the modified master file.

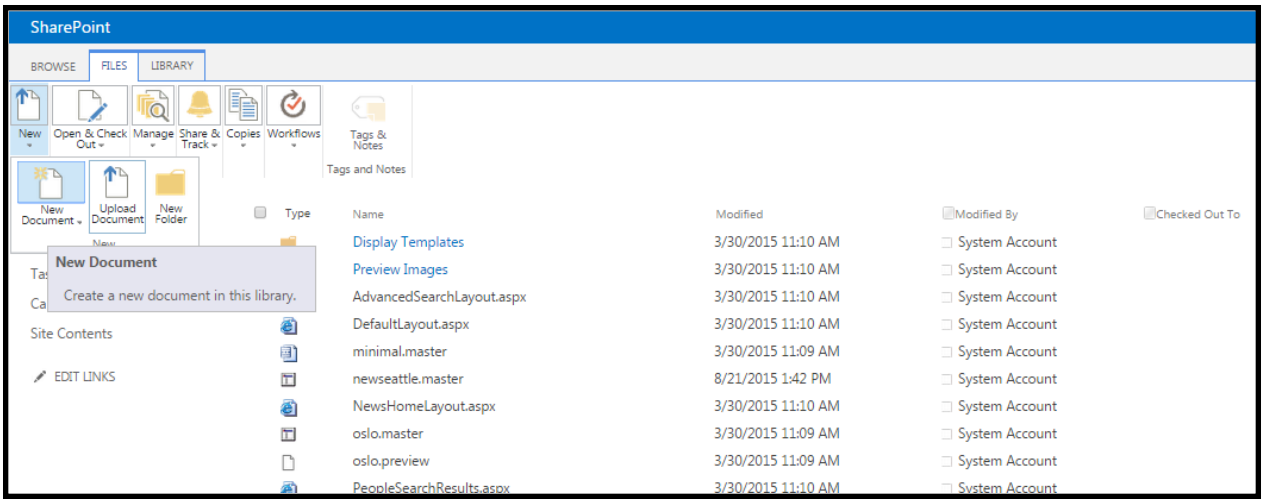


Figure 3.62: Uploading the modified master file

14. Figure 3.63 will then appear. Use the **Choose File** button in Figure 3.63 to select the master file to be uploaded. Provide appropriate comments in the **Version Comments** text box, and click the **OK** button.

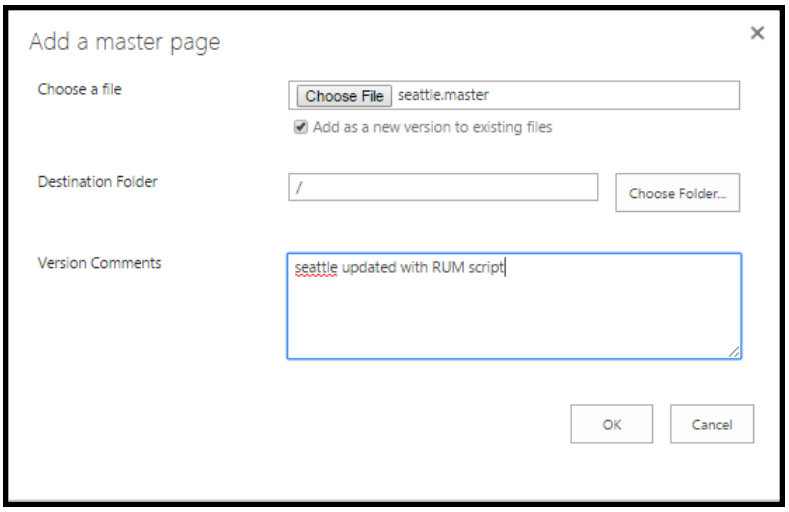


Figure 3.63: Choosing the modified master file for uploading

15. When Figure 3.64 appears, provide a **Description** and click the **Save** button therein.

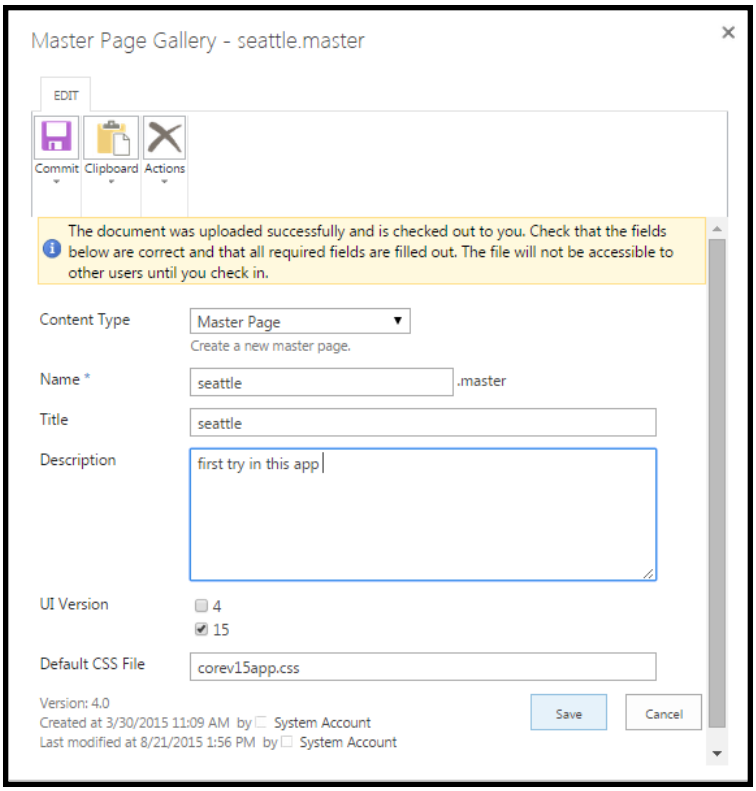


Figure 3.64: Saving the changes to the master file

16. Once back in the **Master Page Gallery**, make sure that the timestamp displayed against the master file reflects the time at which the file was modified (see Figure 3.65).

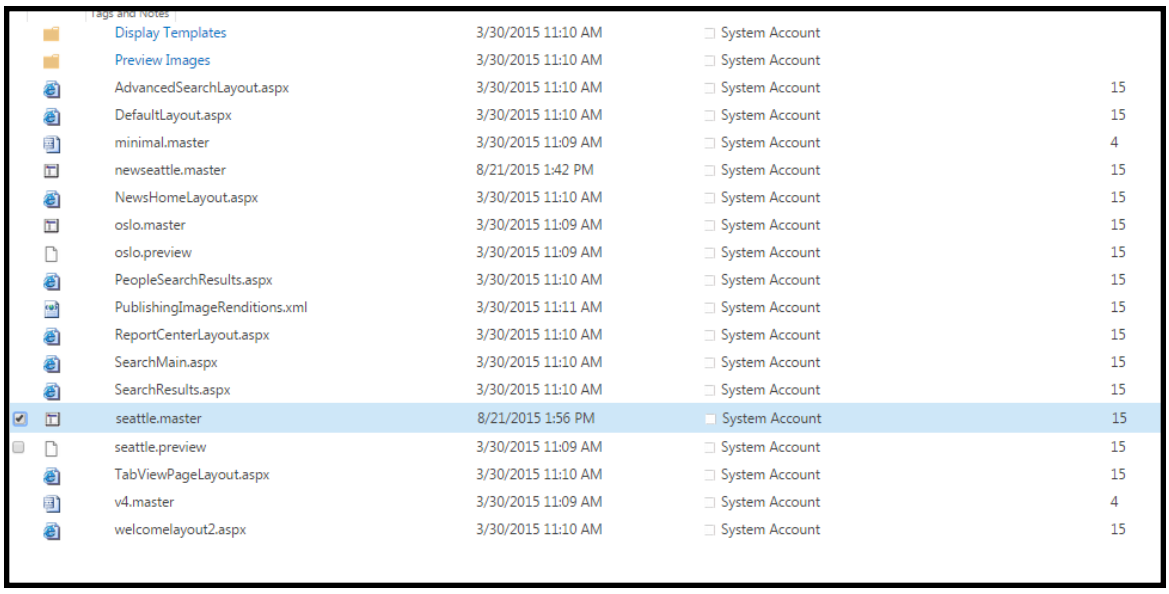


Figure 3.65: Confirming that the timestamp of the master file reflects the time of modification

17. Now, go to Sharepoint's **Central Administration** site, and select the **Manage web applications** option from the **Application Management** section of the site (see Figure 3.66).

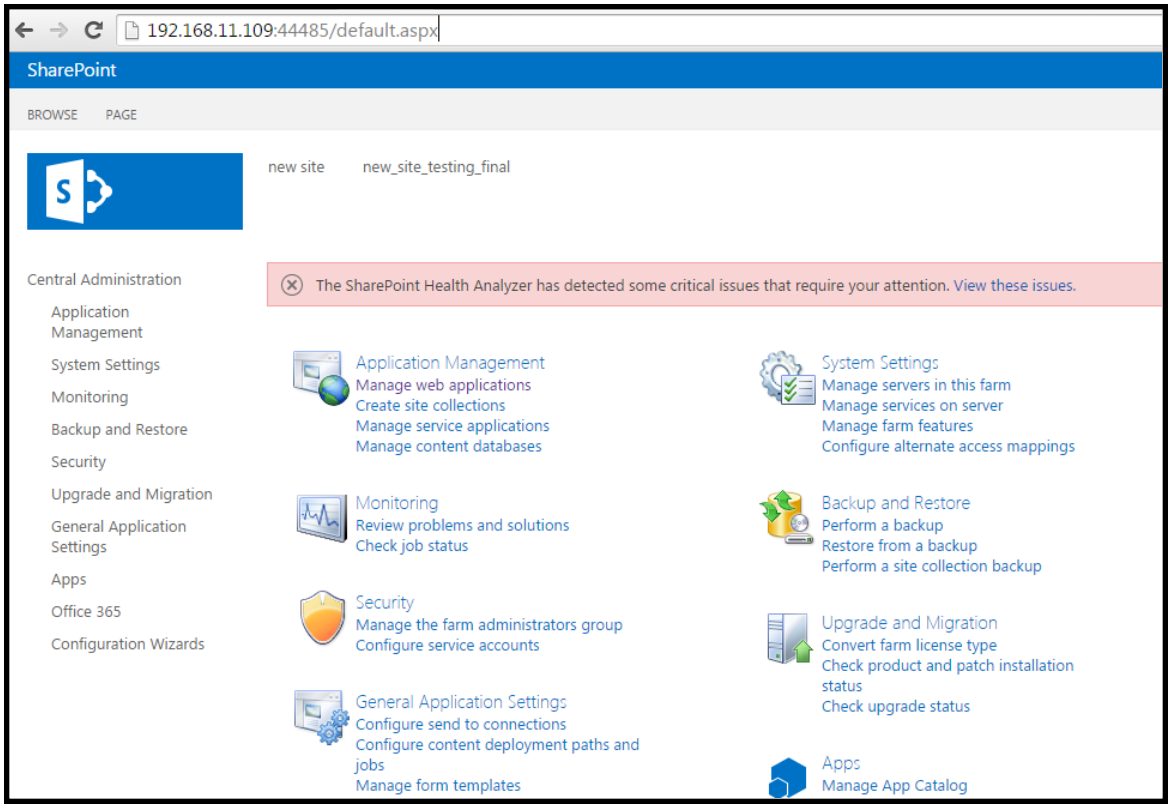


Figure 3.66: Choosing to manage web applications

18. When 3.4.4 appears, select the Sharepoint web application into which you have injected the Javascript code snippet.

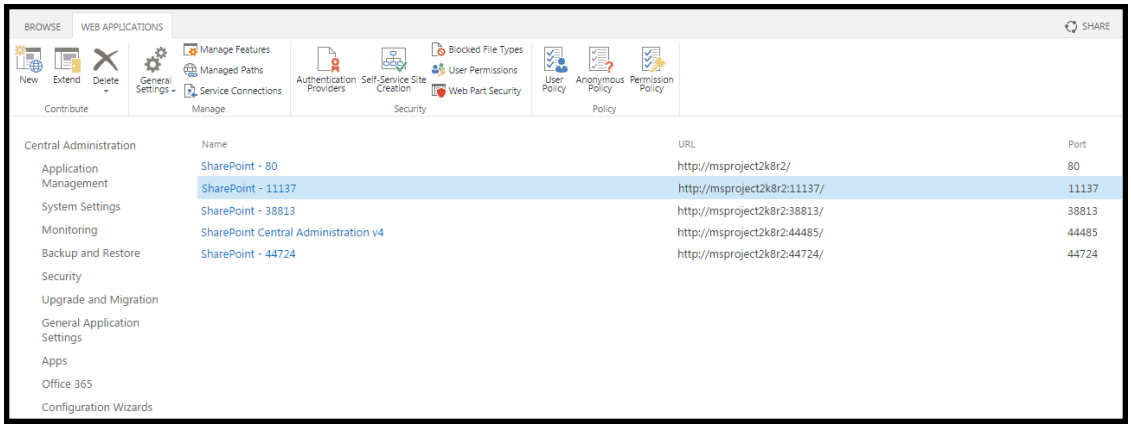


Figure 3.67: Selecting the Sharepoint web application to be monitored

19. From the tool bar at the top of Figure 3.67, click on the **General Settings** tool and then pick the **General**

Settings option from the menu that appears (see Figure 3.68).

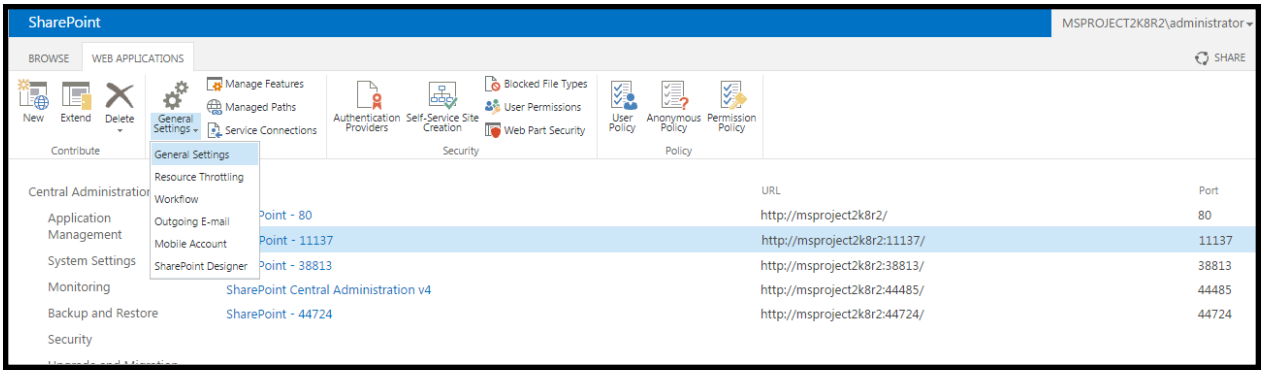


Figure 3.68: Selecting the 'General Settings' option

20. Once Figure 3.69 appears, scroll down the window until you see the **Master Page Setting for Application _Layouts Pages** section. In that section, check whether the **Application _Layouts pages reference site master pages** flag is set to **Yes**. If not, set it to **Yes**.

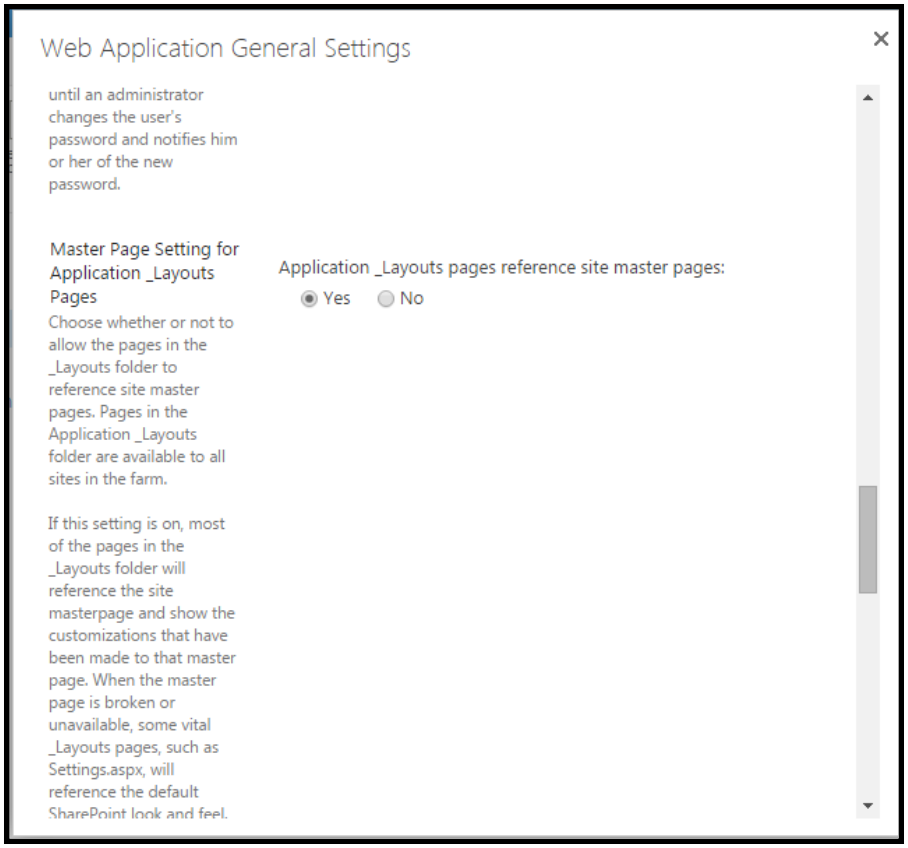


Figure 3.69: Setting the 'Application –Layouts . . .' flag to Yes

Note:

The illustrated example above takes the case of a web application on Sharepoint 2013 that is to be monitored by eG RUM. This is why, the Figures used in the example refer to the **seattle.master** file. When RUM-enabling a web application on Sharepoint 2010, make sure you insert the script in the **v4.master** file. In other words, ensure that step 7 to step 16 of the procedure is performed on the **v4.master** file only, in the case of Sharepoint 2010.

3.4.5 Instrumenting PeopleSoft

Since the PeopleSoft user interface is a combination of HTML, CSS, and JavaScript, it can be easily RUM-enabled. However, PeopleSoft has multiple pages, and it is not advisable to inject the JavaScript into every single page. This is why, prior to instrumenting PeopleSoft, you need to identify an **Application Designer Managed Object** that can act as a vehicle for carrying the eG RUM JavaScript to the client browser. Once such an object is identified, follow the steps below to RUM-enable PeopleSoft:

1. Login to PeopleTools (see Figure 3.70).

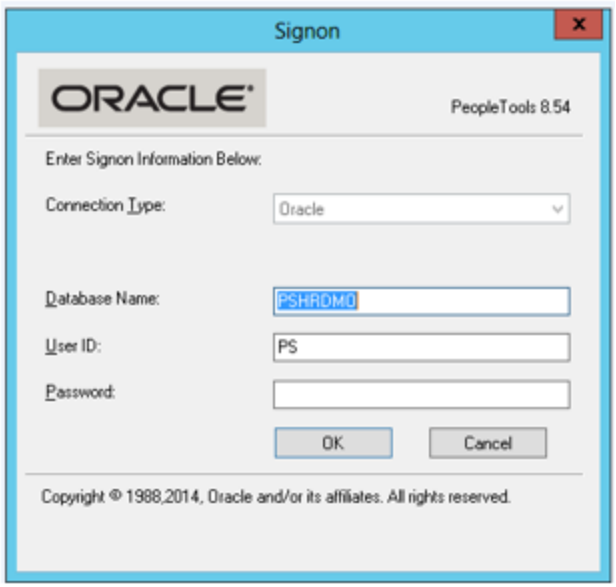


Figure 3.70: Logging into PeopleTools

2. Open the Application Designer. When Figure 3.71 appears, select **HTML** from the **Definitions** drop-down. If you are using PeopleSoft 8.53 or higher, then, in the **Selection Criteria** section, enter **PT_COMMON** in the **Name** text box to locate a definition with that name. If you are using a PeopleSoft version lower than 8.53, then, enter **PT_COPYURL** in the **Name** text box.

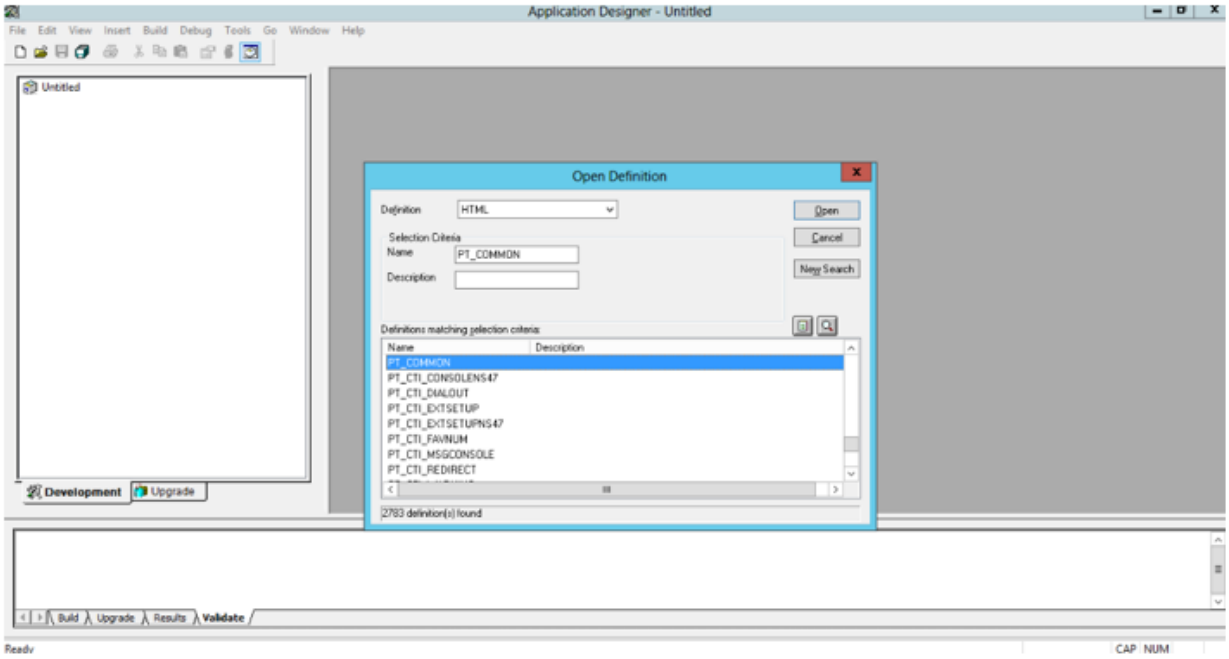


Figure 3.71: Searching for an HTML definition

3. The definitions that match the specified criteria will then be listed in the **Definitions matching selection criteria** list box (see Figure 3.71). Select the **PT_COMMON** (or **PT_COPYURL**, as the case may be) definition from the list and edit it.
4. Figure 3.72 will then appear. Here, paste the JavaScript code snippet as depicted.

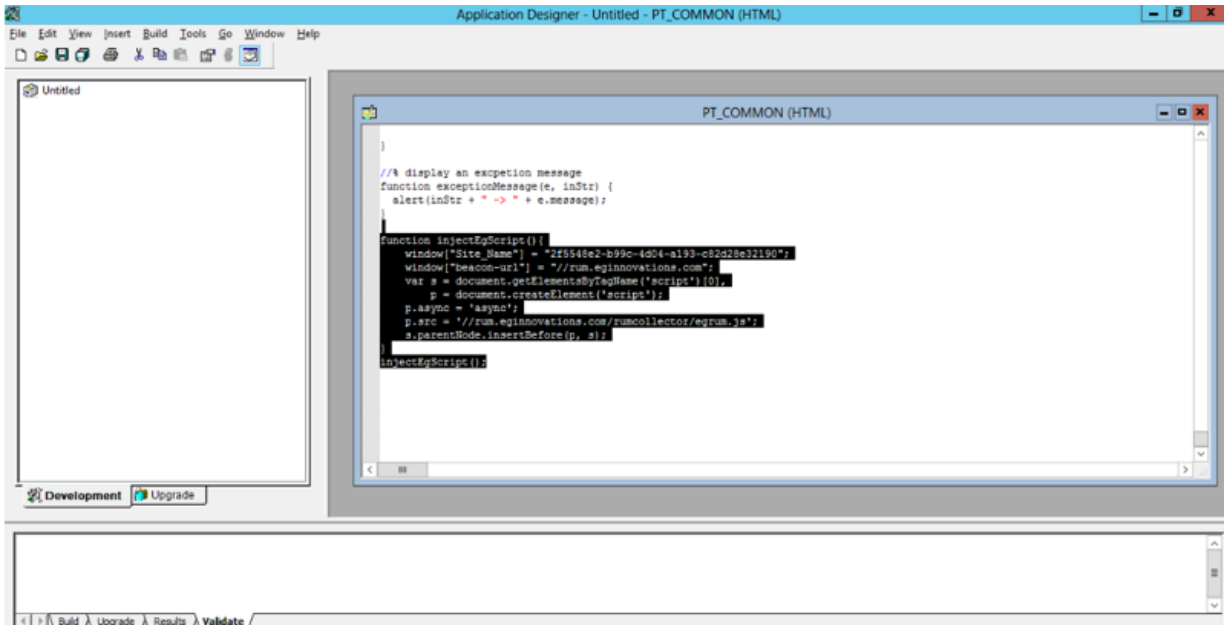


Figure 3.72: Inserting the JavaScript code snippet in the HTML definition

5. Finally, save the changes.

3.4.6 Instrumenting Using the JavaScript Code Snippet

As already mentioned, where web pages are not dynamically generated by the target web site / web application, you can directly copy the JavaScript code snippet that eG provides (in Figure 3.73) to every web page you want monitored.

For that, do the following:

1. Open the html/jsp/aspx web page to be monitored in an Editor.

2. Insert the copied code snippet in the *head* tag of the web page, as depicted by Figure 3.73 below.

Figure 3.73: Inserting the code snippet in a web page

3. Finally, save the edited web page.

3.5 Modifying the Test Configuration

Once the code snippet is inserted into a web page, every time a user loads that page onto his/her browser – i.e., accesses that page via a browser – the browser will run the code snippet, which in turn will download and execute the **egrum.js** Java script. The script will capture the load time of the page and Java script errors that the page encounters. The browser will then send these user experience metrics to the RUM collector.

To gather the metrics reported to the RUM collector, the eG agent periodically runs tests on the data collector. By default, these tests are pre-configured with the following:

- Test frequency;
- Proxy server details
- How many page views are to be tracked?

- URL and Javascript error patterns that are to be excluded from monitoring;
- Slow transaction cut off
- The depth of metrics to be collected as part of detailed diagnosis

If required, you can fine-tune the test configuration to suit your needs. For instance, you can change the test frequency, increase the number of page views you want tracked, exclude more URLs from monitoring, etc. For this purpose, you will have to reconfigure the tests.

To reconfigure a test, follow the steps below:

1. Select the **Specific Configuration** option from the **Tests** menu in the **Agents** tile.
2. Select **Real User Monitor** as the **Component type** and pick the RUM component being managed from the **Component name** drop-down.
3. Then, from the **CONFIGURED TESTS** list, select the test to be modified and click the **Reconfigure** button.
4. The parameters of the chosen test will then appear. You can make the changes you require, and click the **Update** button to save the changes. Detailed explanation for each of the parameters and how to modify them is available in Section **3.6**.

3.6 Monitoring using the eG Real User Monitor

The eG agent will then run the tests at the configured frequencies to pull the web site-related metrics from the RUM collector. These metrics are then routed to the eG manager, which presents them in the eG monitoring console using a specialized monitoring model (see Figure 3.74).

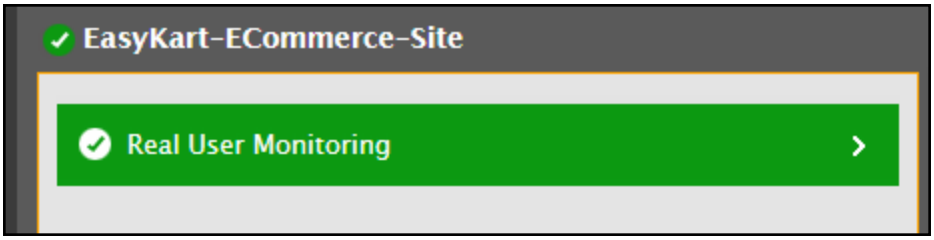


Figure 3.74: Layer model of the eG RUM component

All the tests that the eG agent runs on the collector are grouped under a single **Real User Monitoring** layer of this logical model. Clicking on the **>** button in Figure 3.74, will reveal these tests.

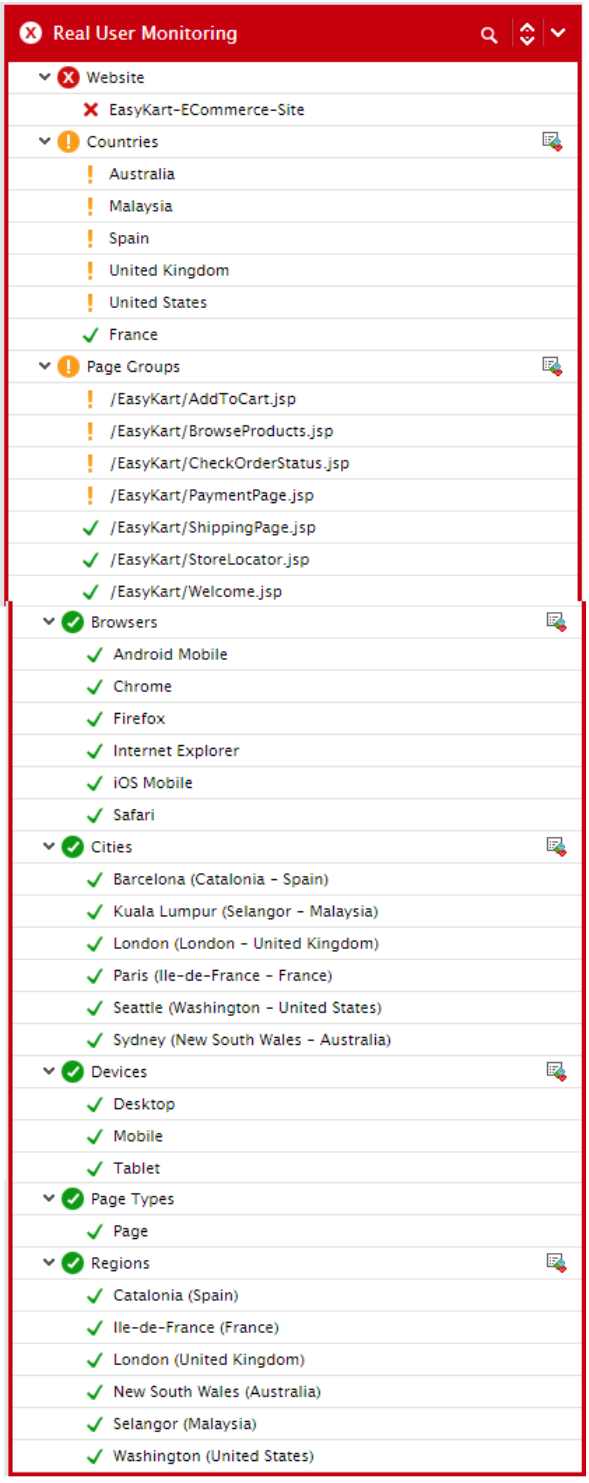


Figure 3.75: Tests mapped to the Real User Monitoring layer

Each of these tests and the measures they report have been discussed elaborately in the sections that follow.

3.6.1 Web Site Test

User experience with a web site is a key measure of web site performance. Typically, the following factors differentiate a 'satisfactory' user experience from a 'slow' or 'frustrating' experience:

- Slow page views;
- Javascript errors in pages;

When users complain of poor experience with a web site, administrators must first have to determine which of the aforesaid factors is adversely impacting user experience – slow pages? error pages? or both? But, this knowledge is not enough to help administrators undo the damage to revenue and reputation that user experience issues cause. For that, administrators also need to know exactly "why" and "where" the slowness occurred and "what" caused the Javascript error, so that they can quickly initiate measures to eliminate the bottleneck and ensure that user experience improves rapidly and significantly!

The challenge in isolating the root-cause of slow page views particularly, is the multi-tier nature of the page loading process. Figure 3.76 below depicts how a web page loads.

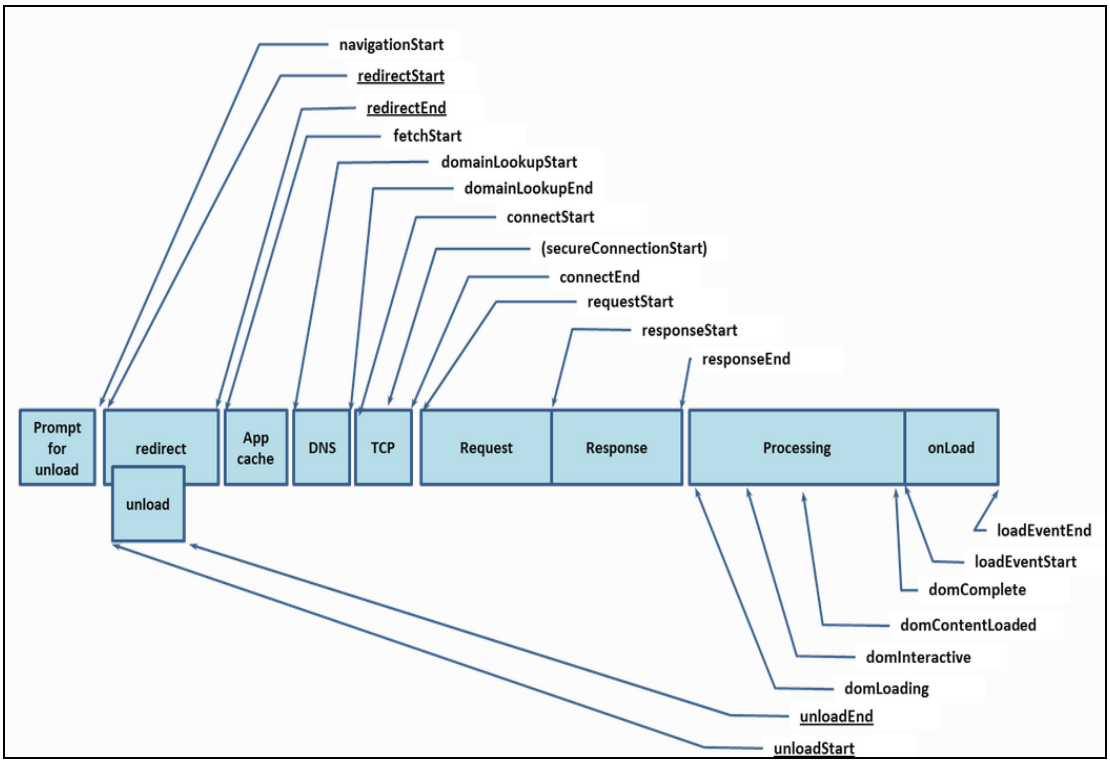


Figure 3.76: The page loading process

As is evident from Figure 3.76, when a user requests for a web page by hitting the URL, the browser first unloads the page (if any) already loaded on to it. Then, the browser checks whether the request needs to be redirected to a different URL. If so, it redirects the request to the other URL. Next, the browser searches the App Cache for resources to be loaded to the requested page. If one/more resources are available in the cache, the browser loads the resources and then attempts to connect to the web/web application server that hosts

the web site for fetching the requested page. When connecting to the server, the browser first looks up the DNS to resolve the server domain name to its IP address. If domain resolution is successful, the browser then establishes a TCP connection with the server via its IP address and port and transmits the page request to the server via that connection. The server then responds with the requested page. Upon receipt of a response from the server, the client browser builds the contents of the page by loading the document object model, and finally renders the page.

A slowdown in any one event that is part of the page loading process can affect the loading time of the page – for instance, if domain lookup takes longer than usual, page load time will increase. This is why, administrators often struggle to precisely pinpoint where the bottleneck is!

This is where the **Web Site** test helps. This test tracks page view requests to a monitored web site/web application, and measures the time taken by the requested pages to load. Administrators are promptly alerted if the load time slides below configured thresholds. Under such circumstances, the test supplements the administrator's troubleshooting efforts by additionally capturing and reporting the time taken by every step of the page loading process. This enables administrators to instantly and accurately isolate the exact step at which the slowdown may have occurred – during domain lookup? When connecting to the server via TCP? when processing the request at the server end? When loading the document? Or when rendering the page? Moreover, the detailed diagnosis of this test accurately pinpoints the slow pages (in terms of load time) and also provides the load time breakup per page, using which administrators can quickly and precisely identify which page is slow and why! The test also reports the count of pages with Javascript errors and provides detailed diagnostics that reveal what these errors are. This way, the test proactively alerts administrators to issues that may potentially affect user experience with a web site, pinpoints the root-cause of such issues, and thus helps administrators take corrective action, well before users notice.

Note:

- The metrics reported by the **Web Site** test forms the basis for the execution of all other tests mapped to the *Real User Monitor* component. So, if the **Web Site** test is excluded or disabled, then none of the other tests will run.
- The **Web Site** test tracks requests to only the base pages of a monitored web site/web application; not to the AJAX or iFrame pages that the web application may support. To capture the responsiveness of requests to AJAX and iFrame pages, you need to use the **Page Types** test instead.

Target of the test : A web site/web application managed as a *Real User Monitor*

Agent deploying the test : A remote agent

Outputs of the test : One set of results for each browser used for accessing the web site/web application being monitored

Test parameters:

TEST PERIOD	How often should the test be executed
PROXYHOST	If the eG agent communicates with the RUM collector via a proxy, then specify the IP address/fully-qualified host name of the proxy server here. By default, this is set to *none*, indicating that the eG agent does not communicate with the collector via a proxy.
PROXYPORT	If the eG agent communicates with the RUM collector via a proxy, then specify the port at which the proxy server listens for requests from the eG agent. By default, this is set

	to *none*, indicating that the eG agent does not communicate with the collector via a proxy.
PROXYUSERNAME and **PROXYPASSWORD**	If the eG agent communicates with the RUM collector via a proxy server, and if proxy server requires authentication, then specify valid credentials for authentication against **PROXYUSERNAME** and **PROXYPASSWORD**. If no proxy server is used or if the proxy server used does not require authentication, then set the **PROXYUSERNAME** and **PROXYPASSWORD** to *none*.
CONFIRM PASSWORD	Confirm the **PROXYPASSWORD BY RETYPING IT HERE.** **Note:** If you **Reconfigure** the test later to change the values of the **PROXYUSERNAME, PROXYPASSWORD**, and **CONFIRM PASSWORD** parameters, then such changes will be effected only if the eG remote agent monitoring the Real User Monitor component is restarted.
DO YOU WANT TO LIMIT THE PAGE VIEWS?	By default, the eG RUM monitors all requests to a managed web site. This is why, this flag is set to **No** by default. However, in case of web sites that receive thousands of hits every day, monitoring each page view may add significantly to the overhead of the eG agent and may also increase the size of the eG database considerably. To reduce the strain on both the eG agent and the eG backend, you may want to restrict the monitoring scope of this test to a few page visits. To achieve this, first set this flag to **Yes**. This will invoke the option depicted by the below figure.  Figure 3.77: Configuring the number of allowed page visits By default, the **MAXIMUM ALLOWED PAGE VISITS PER DAY IS SET TO** *100000*. This implies that the test will consider only the first *100000* requests in a day for monitoring. All page visits beyond *100000* will by default be excluded from the test's monitoring purview. You can increase or decrease this limit, if you so need.
URL SEGMENTS TO BE USED AS GROUPED URL	**This parameter is applicable to the Page Groups test alone. The Page Groups test groups URLs based on the URL segments configured for monitoring and reports aggregated response time metrics for every group.** Using this parameter, you can specify a comma-separated list of URL segment numbers based on which the pages are to be grouped. URL segments are the parts of a URL (after the base URL) or path delimited by slashes. So if you had the URL: http://www.eazykart.com/web/shopping/login.jsp , *then*http://www.eazykart.com will be the base URL or domain, */web will be the first URL segment, /shopping will be the second URL segment, and /login.jsp will be the third URL segment.* By default, this parameter is set to 1,2. This default setting, when applied to the sample URL provided above, implies that the eG agent will aggregate request and response time metrics to all instrumented web pages (i.e., web pages with the code snippet)

	under the URL */web/shopping. Here,* /web corresponds to the specification 1 (URL segment 1) and /shopping corresponds to the specification 2 (URL segment 2) in the default value 1,2. This in turn means that, if the web site consists of pages such as http://www.eazykart.com/web/shopping/products.jsp , http://www.eazykart.com/web/shopping/products/travel/bags.jsp , http://www.eazykart.com/web/shopping/payment.jsp, etc., then the eG agent will track the requests to and responses from all these web pages, aggregate the results, and present the aggregated metrics for the descriptor */web/shopping*. This way, the test will create different page groups based on each of the second-level URL segments in the managed web site – eg., */web/movies, /web/travel, /web/reservations, /partner/contacts etc.* - and will report aggregated metrics for each group so created. If you want, you can override the default setting by providing different URL segment numbers here. For instance, your specification can be just *2. In this case, for the URL* http://www.eazykart.com/web/shopping/login.jsp, the test will report metrics for the descriptor */shopping. You can even set this parameter to 1,3*. For a web site that contains URLs such as http://www.eazykart.com/web/shopping/products.jsp , http://www.eazykart.com/web/shopping/products/travel/bags.jsp , and http://www.eazykart.com/web/shopping/payment.jsp, this specification will result in the following descriptors: */web/products*, */web/products.jsp*, and */web/payment.jsp*.
URL PATTERNS TO BE IGNORED FROM MONITORING	By default, this test does not track requests to the following URL patterns: **.js,*.css,*.jpeg,*.jpg,*.png*. If required, you can remove one/more patterns from this default list, so that such patterns are monitored, or can append more patterns to this list in order to exclude them from monitoring. For instance, to additionally ignore URLs that end with *.gif* and *.bmp* when monitoring, you need to alter the default specification as follows: **.js,*.css,*.jpeg,*.jpg,*.png,*.gif,*.bmp* **Note:** The URL patterns configured here are not just used by the eG agent to filter out unimportant performance data during metrics collection; these patterns are also used by the RUM collector to determine which beacons should be accepted and which ones need to be discarded. Typically, the very first time this test runs and polls the RUM collector for metrics, the eG agent executing this test searches the performance records stored in the collector for data that pertains to the URL patterns configured for exclusion. If such data is found, the agent then ignores that data during metrics collection. This means that during the very first test execution, URL filtering is performed only by the eG agent. During this time, the RUM collector downloads the URL patterns configured against this parameter from the eG agent. Armed with this information, the collector then scans all beacons that browsers send subsequently, and determines if there are any beacons for the excluded URL patterns. If such beacons are found, the collector discards them instantly. Filtering URLs at the collector-level significantly reduces the load on the collector and conserves storage space on the collector; it also minimizes the workload of the eG agent. By additionally filtering URLs at the agent-level, eG makes sure that even if beacons pertaining to excluded URL patterns find their way into the RUM

	collector, they are captured and siphoned out by the eG agent.
JAVASCRIPT ERRORS TO BE IGNORED	By default, this test alerts administrators to all Javascript errors that occur in the monitored web site/web application. This is why, this parameter is set to *none* by default. Sometimes however, administrators may not want to be notified when certain types of Javascript errors occur – this could be because such errors are harmless or are a normal occurrence in their environment. In such circumstances, you can instruct the eG agent to ignore these errors when monitoring. For this, specify the Java script error message to be ignored in the **JAVASCRIPT ERRORS TO BE IGNORED** text box, in the following format: *<Javascript eror message>:<URL of the page/file where the message originated>.* For instance, say that the *login.html* page in your web site runs a few Java scripts that throw *Object expected errors*, which you want the eG agent to ignore. In this case, your error specification can be as follows: *Object expected:http://www.eazykart.com/web/login.html*. Alternatively, you can provide only that text string with which the error message begins in your specification – eg., *Object:http://www.eazykart.com/web/login.html*. Moreover, instead of the complete URL, you can specify just the name of the HTML/jsp/aspx page in which the error is to be ignored – example: *Object:login.html*. Sometimes, the individual web pages in your web site may not run any Java script directly. Instead, these web pages may include links to Java script files that will run the Java script and return the output to the web pages. If you want the eG agent to ignore certain errors thrown by such a Javascript file, then your error pattern specification should include the URL of the Javascript file and not the web page that points to it. This is because, in this case, the file is where the error message originates. For instance, in the same example above, if the *login.html* page points to a *validate.js* file, and you want to ignore the Object expected errors that this JS file throws, your error pattern specification will either be, *Object expected:validate.js*, or *Object:validate.js*. Multiple error message-URL combinations can also be provided as a comma-separated list. The format of your specification will be: *<Javascript error message 1>:<Originating URL 1>,<Javascript error messa ge 2>:<Originating URL 2>,. . .* For example, to ignore the *Object expected and Uncaught TypeError errors in the login.html page, use the following specification:* *Object:login.html,Uncaught:login.html* Likewise, to ignore the *Object expected error in the login.html page and the Uncaught TypeError in the validate.js file, your specification will be:* *Object:login.html,Uncaught:validate.js* If you want to ignore the *Uncaught TypeError* across all the pages of the web site, your specification will be as follows:

	Uncaught TypeError:All **Note:** When specifying the *<Javascript error message>* to be ignored, take care of the following: • The error message should not contain any special characters – in particular, the ':' (the colon) and the ',' (the comma) characters should be avoided. • The case of the actual error message and the one specified as part of your error specification should match. This is because, the eG agent performs *case-sensitive* pattern matching.
MAXIMUM HEALTHY TRANSACTIONS IN DD	By default, this parameter is set to *5* indicating that the detailed diagnosis of this test will collect and display metrics related to the top-5 normal/healthy page views, in terms of the Average page load time. To identify the top 5, the eG agent sorts all healthy transactions in the ascending order of their Average page load time, and picks the first *5* transactions from this sorted list for display in the **DETAILED DIAGNOSIS** page and for storing in the eG database. You can however, increase or decrease this value depending upon how many healthy transactions you want to see in your detailed diagnosis, and how well-tuned or well-sized the eG database is. If you do not want the detailed diagnosis to include any healthy transaction, set the value of this parameter to *0*. To view all healthy transactions, set the value of this parameter to *all*. Before setting the value of this parameter to *all*, make sure that you have a well-sized eG database in place, as this setting will store details of every transaction that registers an *Avg page load time* value that is lower than the **SLOW TRANSACTION CUTOFF** specification. On a good day, this can result in numerous transactions, and can consume considerable space in the eG database.
MAXIMUM SLOW TRANSACTIONS IN DD	By default, this parameter is set to *5* indicating that the detailed diagnosis of this test will display metrics related to all the top-5 slow transactions, in terms of the *Average page load time*. To identify the top 5, the eG agent sorts all slow transactions in the descending order of their *Average page load time*, and picks the first *5* transactions from this sorted list for display in the **DETAILED DIAGNOSIS** page and for storing in the eG database. You can however, increase or decrease this value depending upon how many slow transactions you want to see in your detailed diagnosis, and how well-tuned or well-sized the eG database is. If you do not want the detailed diagnosis to include any slow transaction, set the value of this parameter to *0*. To view all slow transactions, set the value of this parameter to *all*. Before setting the value of this parameter to *all*, make sure that you have a well-sized eG database in place, as this setting will store details of every transaction that registers an *Avg page load time* value that is higher than the slow transaction cutoff specification. On a bad day, this can result in numerous transactions, and can consume considerable space in the eG database.

MAXIMUM ERROR TRANSACTIONS IN DD	By default, this parameter is set to *5*, indicating that the detailed diagnosis of this test will display metrics related to the top-5 transactions that encountered JavaScript errors, based on when those errors occurred. To identify the top 5, the eG agent sorts all error transactions in the descending order of the date/time at which the errors occurred, and picks the first 5 transactions from this sorted list for display in the detailed diagnosis page and for storing in the eG database. You can however, increase or decrease this value depending upon how many error transactions you want to see in your detailed diagnosis, and how well-tuned or well-sized the eG database is. If you do not want the detailed diagnosis to include any error transaction, set the value of this parameter to *0*. To view all error transactions, set the value of this parameter to *all*. Before setting the value of this parameter to *all*, make sure that you have a well-sized eG database in place, as this setting will store details of every transaction that encounters a JavaScript error. On a bad day, this can result in numerous transactions, and can consume considerable space in the eG database.
SLOW TRANSACTION CUTOFF (MS)	This test reports the count of slow page views and also pinpoints the pages that are slow. To determine whether/not a page is slow, this test uses the **SLOW TRANSACTION CUTOFF** parameter. By default, this parameter is set to *4000 millisecs (i.e., 4 seconds)*. This means that, if a page takes more than 4 seconds to load, this test will consider that page as a slow page by default. You can increase or decrease this slow transaction cutoff according to what is 'slow' and what is 'normal' in your environment. **Note:** The default value of this parameter is the same as the default *Maximum threshold* setting of the *Avg page load time* measure – i.e., both are set to *4000 millisecs* by default. While the former helps eG to distinguish between slow and healthy page views for the purpose of providing detailed diagnosis, the latter tells eG when to generate an alarm on *Avg page load time*. For best results, it is recommended that both these settings are configured with the same value at all times. Therefore, if you change the value of one of these configurations, then make sure you update the value of the other as well. For instance, if the **SLOW TRASACTION CUTOFF IS CHANGED TO** *6000 millisecs, change* the *Maximum Threshold* of the *Avg page load time* measure to *6000* millsecs as well.
PAGELOADTIME CUTOFF (MS)	eG RUM relies on the Navigation API to track page views and capture page load time. Sometimes, the Navigation API can incorrectly report abnormally high values for page load time. To ensure that the eG agent ignores beacons with such page load time values, you can configure in this text box, the maximum value for page load time. By default, this is set to 3600000 (ms). This implies that by default, the eG agent will disregard all beacons that carry page load time values higher than 3600000 milliseconds. You can change this value to suit your environment.
PAGE TYPES TO BE INCLUDED IN DASHBOARD	By default, the eG RUM Dashboard displays details of base page views only. You can optionally include Ajax and/or iFrame views as well in the dashboard, by selecting the relevant options from this list box.

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**.
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.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Page views:	Indicates the total number of times pages in this web site/web application were viewed by users.	Number	This is a good measure of the traffic to your web site/web application, and also reveals how popular your web site is. An unusually high number of page views could be a cause for concern, as it could be owing to a malicious virus attack or an unscrupulous attempt to hack your web site/web application. Either way, be wary of sudden, but significant spikes in the page view count! **Note:** An abnormally high value for this measure may not always be a cause for concern; nor would it always indicate a genuine increase in traffic to the web site/web application. If the eG agent monitoring the *Real User Monitor* component is stopped for

Measurement	Description	Measurement Unit	Interpretation
			sometime (say, for maintenance purposes) and then started, or if the eG agent-collector connection breaks and is restored after a while, the eG agent will pull all the metrics that the collector stored locally during the period of its absence, cumulate them, and then display the cumulated values in the eG monitoring console as metrics that pertain to the current measurement period. In reality, these metrics pertain to the entire time period when the eG agent was unavailable. Because of this, the *Page views* measure may indicate a sudden and significant surge in traffic.
Apdex score:	Indicates the apdex score of this web site/web application.	Number	Apdex (Application Performance Index) is an open standard developed by an alliance of companies. It defines a standard method for reporting and comparing the performance of software applications in computing. Its purpose is to convert measurements into insights about user satisfaction, by specifying a uniform way to analyze and report on the degree to which measured performance meets user expectations. The Apdex method converts many measurements into one number on a uniform scale of 0-to-1 (0 = no users satisfied, 1 = all users satisfied). The resulting Apdex score is a numerical measure of user satisfaction with the performance of enterprise applications. This metric can be used to report on any source of end- user performance measurements for which a performance objective has been defined. The Apdex formulais:

Measurement	Description	Measurement Unit	Interpretation
			Apdex $_t$ = (Satisfied Count + Tolerating Count / 2) / Total Samples This is nothing but the number of satisfied samples plus half of the tolerating samples plus none of the frustrated samples, divided by all the samples. A score of 1.0 means all responses were satisfactory. A score of 0.0 means none of the responses were satisfactory. Tolerating responses half satisfy a user. For example, if all responses are tolerating, then the Apdex score would be 0.50. Ideally therefore, the value of this measure should be *1.0*. A value less than *1.0* indicates that the user experience with the web site/web application has been less than satisfactory.
Average page load time:	Indicates the average time taken by the pages in this web site to load completely.	ms	This is the average interval between the time that a user initiates a request and the completion of the page load of the response in the user's browser. In the context of an Ajax request, it ends when the response has been completely processed. If the value of this measure is consistently high for a web site/ web application, there is reason to worry. This is because, it implies that the web site/web application is slow in responding to requests. If this condition is allowed to persist, it can adversely impact user experience with the web site/web application. You may want to check the *Apdex score* in such circumstances to determine whether/not user experience has already been affected. Regardless,

Measurement	Description	Measurement Unit	Interpretation
			you should investigate the anomaly and quickly determine where the bottleneck lies – is it at the front end? network? or backend? - so that the problem can be fixed before users even notice any slowness! For that, you may want to compare the values of the *Average front end time, Average server connection time, and Average response available time measures of this test.* If the *Average front end time is the highest, it indicates that the problem is with the web site/web application front end – this can be attributed to a slowdown in page rendering or in DOM building. If the Average server connection time* is the highest, it denotes that the network is the problem source. This in turn can be caused by TCP connection latencies and delays in domain look up. On the other hand, if the *Average response available time* measure registers the highest value, it indicates that the problem lies with the web site/web application backend – i.e., the web/web application server that is hosting the web site/web application being monitored. You can also use the detailed diagnosis of this measure to identify the exact pages in the web site/web application that are presently slow. With the help of the detailed metrics provided per page, you can even precisely pinpoint the root-cause of the slowness of that page.
Unique user session:	Indicates the number of distinct users who are currently accessing this web site/web application.	Number	

Measurement	Description	Measurement Unit	Interpretation
Page views per minute:	Indicates the number of times the pages in this web site/web application were viewed per minute.	Number	This is a good indicator of the level of activity on the web site. An unusually high value for this measure may require investigation.
Normal page view percentage:	Indicates the percentage of page views with a normal user experience.	Percent	The value of this measure indicates the percentage of page views in which users have neither experienced any slowness, nor encountered any Javascript errors. Ideally, the value of this measure should be 100%. A value less than 100% indicates the existence of one/more slow/error-prone pages in the web site. A value less than 50% is indicative of a serious problem, where most of the page views are either slow or have encountered Javascript errors. Under such circumstances, to know what exactly is affecting user experience, compare the value of the *Slow page view percentage* with that of the *Javasript error page view percentage*. This will reveal the reason for the poor user experience with the web site/web application – slow pages? or Javascript errors?
Slow page view percentage:	Indicates the percentage of page views that are slow in loading.	Percent	Ideally, the value of this measure should be 0. A value over 50% implies that you are in a spot of bother, with over half of the page views being slow. Use the detailed diagnosis of the *Slow page views* measure to identify the slow pages and isolate the root-cause of the slowness – is it a slow frontend? bad network? or a malfunctioning backend?
JavaScript error view percentage:	Indicates the percentage of page views that have encountered JavaScript	Percent	Ideally, the value of this measure should be 0. A value over 50% implies that you are in a spot of bother, with over half of

Measurement	Description	Measurement Unit	Interpretation
	errors.		the page views experiencing JavaScript errors.
Slow page views (Tolerating & Frustrated):	Indicates the number of times pages in this web site/web application took very long to be viewed.	Number	A page view is considered to be slow when the average time taken to load that page exceeds the **SLOW TRANSACTION CUTOFF CONFIGURED FOR THIS TEST.** Ideally, a page should load quickly. The value 0 is hence desired for this measure. If the value of this measure is high, it indicates that users frequently experienced slowness when accessing pages in the web site/web application. To know which page views are slow and why, use the detailed diagnosis of this measure.
JavaScript error page views:	Indicates the number of times JavaScript errors occurred when viewing the pages in this web site/web application.	Number	Ideally, the value of this measure should be 0. A high value indicates that many JavaScript errors are occurring when viewing pages in the web site/web application. Use the detailed diagnosis of this measure to identify the error pages and to know what Javascript error has occurred in which page. This will greatly aid troubleshooting!
Satisfied page views:	Indicates the number of times pages were viewed in the web site without any slowness.	Number	A page view is considered to be slow when the average time taken to load that page exceeds the **SLOW TRANSACTION CUTOFF** configured for this test. If this **SLOW TRANSACTION CUTOFF** is not exceeded, then the page view is deemed to be 'satisfactory'. To know which page views are satisfactory, use the detailed diagnosis of this measure. Ideally, the value of this measure should be the same as that of the *Page views* measure. If not, then it indicates that

Measurement	Description	Measurement Unit	Interpretation
			one/more page views are slow – i.e., have violated the **SLOW TRANSACTION CUTOFF**. If the value of this measure is much lesser than the value of the *Tolerating page* views and the *Frustrated page views*, it is a clear indicator that web site performance is below-par. In such a case, use the detailed diagnosis of the *Tolerating page views* and *Frustrated page views* measures to know which pages are slow and why.
Tolerating page views:	Indicates the number of tolerating page views to the web site/web application.	Number	If the Average page load time of a page exceeds the **SLOW TRANSACTION CUTTOFF** configuration of this test, but is less than 4 times the **SLOW TRANSACTION CUTOFF** (i.e., < 4 * slow transaction cutoff), then such a page view is considered to be a *Tolerating page view*. Ideally, the value of this measure should be 0. A value higher than that of the *Satisfied page views* measure is a cause for concern, as it implies that the overall user experience with the pages in the web site is less than satisfactory. To know which pages are contributing to this sub-par experience, use the detailed diagnosis of this measure. The detailed metrics will also enable you to accurately isolate what is causing the tolerating page views – a problem with the frontend? network? or backend?
Frustrated page views:	Indicates the number of frustrated page views to this web site/web application.	Number	If the Average page load time of a page is over 4 times the **SLOW TRANSACTION CUTTOFF** configuration of this test(i.e., > 4 * **SLOW TRANSACTION CUTOFF**), then

Measurement	Description	Measurement Unit	Interpretation
			such a page view is considered to be a Frustrated page view. Ideally, the value of this measure should be 0. A value higher than that of the *Satisfied page views* measure is a cause for concern, as it implies that the overall user experience with the pages in the web site is less than satisfactory. To know which pages are contributing to this sub-par experience, use the detailed diagnosis of this measure. The detailed metrics will also enable you to accurately isolate what is causing the frustrated page views – a problem with the frontend? network? or backend?
Desktop page views:	Indicates the number of times this web site/web applicationwas accessed from client desktops.	Number	To know which pages in the web site were accessed by desktop users and to evaluate the experience of the desktop users with each of these pages, use the detailed diagnosis of this measure. In the process, slow pages can be identified and the reason for the slowness can be pinpointed.
Mobile page views:	Indicates the number of times this web site/web application was accessed from mobile phones.	Number	To know which pages in the web site were accessed by mobile phone users and to evaluate the experience of these users with each of the pages, use the detailed diagnosis of this measure. In the process, slow pages can be identified and the reason for the slowness can be pinpointed.
Tablet page views:	Indicates the number of times this web site/web application was accessed from tablets.	Number	To know which pages in the web site were accessed by tablet users and to evaluate the experience of these users with each of the pages, use the detailed diagnosis of this measure. In the process,

Measurement	Description	Measurement Unit	Interpretation
			slow pages can be identified and the reason for the slowness can be pinpointed.
Average front end time:	Indicates the interval between the arrival of the first byte of text response and the completion of the response page rendering by the browser.	ms	In the Figure 3.76, that depicts a typical page loading process, the *Average front end time* denotes the time from the response Start event to the load Event End. This process includes document downloading, processing, and page rendering. This time is therefore the sum of the *Average document ready time* and the *Average page rendering time*. If the *Average page load time* of the web site/web application exceeds its threshold, then you may want to compare the value of this measure with that of the *Average server connection time* and *Average response available time* to zoom into the source of the slowness – is it the front end? the network? or the backend? If the *Average front end time* is the highest, it indicates that the problem is with the web site/web application front end - this can be attributed to a slowdown in page rendering or in DOM building. To nail the precise cause of the slowdown, compare the values of the *Average DOM processing time*, *Average DOM download time*, and the *Average page rendering time* measures. On the basis of this comparison, you will be able to tell whether the slowness occurred when downloading the document, when processing it, or when rendering the response page in the browser.
Average page rendering time:	Indicates the time taken to complete the download of	ms	A high value of this measure indicates that the web site/web application is taking

Measurement	Description	Measurement Unit	Interpretation
	remaining resources, including images, and to finish rendering the page.		too long to render the page in the browser. This can adversely impact the *Average front end time*, which in turn can prolong the *Average page load time*. Ideally therefore, the value of this measure should be low.
Average DOM ready time:	Indicates the time taken to make the complete HTML document (DOM) available for JavaScript to apply rendering logic.	ms	The value of this measure is the sum of the *Average DOM download time* and the *Average DOM processing time* measures. If the value of this measure is very high, then you may want to compare the *Average DOM download time* and the *Average DOM processing time* measures to figure out what is delaying DOM building – downloading? Or processing? A high value for this measure can adversely impact the *Average front end time*, which in turn can prolong the *Average page load time*. Ideally therefore, the value of this measure should be low.
Average DOM download time:	Indicates the time taken to download the complete HTML document on the browser.	ms	Higher the download time of the document, longer will be the time taken to make the document available for page rendering. As a result, the overall user experience with the web site/web application will be affected! This is why, a low value is desired for this measure at all times.
Average DOM processing time:	Indicates the time taken to build the Document Object Model (DOM) and make it available for JavaScript to apply rendering logic.	ms	An unusually high value for this measure is a clear indicator that DOM building is taking longer than normal. In consequence, page rendering will be delayed, thus adversely impacting user experience with the web site/web application. Ideally therefore, the value of this measure should be low.

Measurement	Description	Measurement Unit	Interpretation
Average first byte time:	Indicates the interval between the time that a user initiates a request and the time that the browser receives the first response byte. In the context of an Ajax request, this is the interval between the Ajax request dispatch and the time that the browser receives the first response byte.	ms	Going by Figure 3.76 above, the *Average first byte time* is the time that elapsed between navigationStart and responseStart. The value of this measure is also the sum of *Average response available time* , *Average DNS lookup time*, and *Average TCP connection time*. This means that an abnormal increase in any of the above-mentioned time values will increase the value of this measure. If the first response byte from the target web site/web application is itself received slowly, it is bound to have a cascading effect on all events that follow – such as, document downloading, processing, and page rendering. Ultimately, this will impact the page load time as seen by end-users. This is why, if the *Average first byte time* violates its threshold, administrators need to instantly switch to the troubleshooting mode and rapidly isolate what is causing it – is DNS lookup taking a long time? is the network connection to the web site/web application latent? or is the web server/web application server hosting the web site slow in processing requests? By comparing the values of the *Average response available time*, *Average DNS lookup time* , and *Average TCP connection time* measures, administrators can swiftly and accurately figure out the exact reason why there was a delay in receiving the first response byte. If this comparison reveals that the *Average DNS lookup time* is the highest, it implies that domain name resolution by the DNS server is taking a long time and

Measurement	Description	Measurement Unit	Interpretation
			impacting responsiveness. If the *Average TCP connection time* is found to be the culprit, then blame the network connection for delaying the transmission of the response byte. If the *Average reponse available time* is higher than the rest, you can be rest assured that the source of the problem lies with the server hosting the web site/web application.
Average response available time:	Indicates the interval between the start of processing of a request on the browser to when the browser receives the response.	ms	In Figure 3.76, the *Average response available time is the time spent between the requestStart event and responseStart event.* Ideally, a low value is desired for this measure, as high values will certainly hurt the *Apdex scrore of the web site/web application.* The key factor that can influence the value of this measure is the request processing ability of the web server/web application server that is hosting the web site/web application being monitored. Any slowdown in the backend web server/web application server – caused by the lack of adequate processing power in or improper configuration of the backend server - can significantly delay request processing by the server. In its aftermath, the *Average response available time will increase, leaving users with an unsatisfactory experience with the web site/web application.*
Average server connection time:	Indicates the elapsed time since a user initiates a request and the start of fetching the response document from the server or application.	ms	In Figure 3.76, the time spent between *navigationStart and requestStart makes up the Average server connection time. This includes the time to perform DNS lookups and the time to establish a TCP connection with the server. In other words, the value of*

Measurement	Description	Measurement Unit	Interpretation
			this measure is nothing but the sum of the Average DNS lookup time and the Average TCP connection time measures. Ideally, the value of this measure should be low. A very high value will often end up delaying page loading and damaging the quality of the web site service. In the event that the server connection time is high therefore, simply compare the values of the *Average DNS lookup time* and *Average TCP connection time* measures to know to what this delay can be attributed – a delay in domain name resolution? Or a poor network connection to the server?
Average DNS lookup time:	Indicates the time taken by the browser to perform the domain lookup for connecting to this web site/web application.	ms	A high value for this measure will not only affect DNS lookup, but will also impact the *Average server connection time* and *Average page load time* of the web site/web application. This naturally will have a disastrous effect on user experience.
Average TCP connection time:	Indicates the time taken by the browser to establish a TCP connection with the server.	ms	A bad network connection between the browser client and the server can delay TCP connections to the server As a result, the *Average server connection time* too will increase, thus impacting page load time and overall user experience with the web site/web application.

3.6.2 Browsers Test

A robust web site/web application is one that can guarantee a superlative experience for its users, regardless of the browser used. When attempting to build such a web site/web application, developers/designers would typically want to know whether/not user experience varies with browser, and if so, in which way - are specific browsers unable to serve specific pages/page types quickly? are specific browsers more error-prone than the rest? if so, what type of JavaScript errors occur and in which web pages? These insights, if available, are just

the ammunition that developers/designers need to detect and attack browser-compatibility issues, well before the web site/web application goes live! In the absence of these useful inputs at the development stage, compatibility issues come to light only when end-users begin to actively use the web site/application effectively. The result? - prolonged application outages, increased support costs, and many unhappy users! The **Browsers** test seeks to avoid this unpleasant eventuality!

This test automatically discovers the browsers uses by the users to the web site/web application. Then, for every browser so discovered, the test measures and reports the experience of users using that browser. How much time pages accessed using that browser take to load is reported, along with the number of slow pages and their URLs. You can quickly compare these statistics across browsers and figure out whether/not the same pages are slow across browsers. Likewise, you can compare across browsers, the rate at which JavaScript errors occur in the target web site/web application. With the help of the pointers provided by this test, developers/designers can either remove the chinks in the application armour and make it compatible across browsers, or identify the incompatible browsers and advise users against using them to access their application. Either way, users are assured of a good experience with the web site/web application and troubleshooting time and costs are minimized.

Target of the test : A web site/web application managed as a *Real User Monitor*

Agent deploying the test : A remote agent

Outputs of the test : One set of results for each browser used for accessing the web site/web application being monitored

Test parameters:

TEST PERIOD	How often should the test be executed
PROXYHOST	If the eG agent communicates with the RUM collector via a proxy, then specify the IP address/fully-qualified host name of the proxy server here. By default, this is set to *none*, indicating that the eG agent does not communicate with the collector via a proxy.
PROXYPORT	If the eG agent communicates with the RUM collector via a proxy, then specify the port at which the proxy server listens for requests from the eG agent. By default, this is set to *none*, indicating that the eG agent does not communicate with the collector via a proxy.
PROXYUSERNAME and **PROXYPASSWORD**	If the eG agent communicates with the RUM collector via a proxy server, and if proxy server requires authentication, then specify valid credentials for authentication against **PROXYUSERNAME** and **PROXYPASSWORD**. If no proxy server is used or if the proxy server used does not require authentication, then set the **PROXYUSERNAME** and **PROXYPASSWORD** to *none*.
CONFIRM PASSWORD	Confirm the **PROXYPASSWORD** by retyping it here. **Note:** If you **Reconfigure** the test later to change the values of the **PROXYUSERNAME**, **PROXYPASSWORD**, and **CONFIRM PASSWORD** parameters, then such changes will be effected only if the eG remote agent monitoring the Real User Monitor component is restarted.
DO YOU WANT TO	By default, the eG RUM monitors all requests to a managed web site. This is why, this

LIMIT THE PAGE VIEWS?	flag is set to **No** by default. However, in case of web sites that receive thousands of hits every day, monitoring each page view may add significantly to the overhead of the eG agent and may also increase the size of the eG database considerably. To reduce the strain on both the eG agent and the eG backend, you may want to restrict the monitoring scope of this test to a few page visits. To achieve this, first set this flag to **Yes**. This will invoke the option depicted by the below figure.  Figure 3.78: Configuring the number of allowed page visits By default, the **MAXIMUM ALLOWED PAGE VISITS PER DAY** is set to *100000*. This implies that the test will consider only the first 100000 requests in a day for monitoring. All page visits beyond 100000 will by default be excluded from the test's monitoring purview. You can increase or decrease this limit, if you so need.
URL SEGMENTS TO BE USED AS GROUPED URL	**This parameter is applicable to the Page Groups test alone**. The **Page Groups** test groups URLs based on the URL segments configured for monitoring and reports aggregated response time metrics for every group. Using this parameter, you can specify a comma-separated list of URL segment numbers based on which the pages are to be grouped. URL segments are the parts of a URL (after the base URL) or path delimited by slashes. So if you had the URL: http://www.eazykart.com/web/shopping/login.jsp , thenhttp://www.eazykart.com will be the base URL or domain, */web* will be the first URL segment, */shopping* will be the second URL segment, and */login.jsp* will be the third URL segment. By default, this parameter is set to *1,2*. This default setting, when applied to the sample URL provided above, implies that the eG agent will aggregate request and response time metrics to all instrumented web pages (i.e., web pages with the code snippet) under the URL */web/shopping*. Here, */web* corresponds to the specification 1 (URL segment 1) and */shopping* corresponds to the specification 2 (URL segment 2) in the default value *1,2*. This in turn means that, if the web site consists of pages such as http://www.eazykart.com/web/shopping/products.jsp , http://www.eazykart.com/web/shopping/products/travel/bags.jsp , http://www.eazykart.com/web/shopping/payment.jsp, etc., then the eG agent will track the requests to and responses from all these web pages, aggregate the results, and present the aggregated metrics for the descriptor /web/shopping. This way, the test will create different page groups based on each of the second-level URL segments in the managed web site – eg., */web/movies* , */web/travel* , */web/reservations* , */partner/contacts* etc. – and will report aggregated metrics for each group so created. If you want, you can override the default setting by providing different URL segment numbers here. For instance, your specification can be just *2*. In this case, for the URL http://www.eazykart.com/web/shopping/login.jsp, the test will report metrics for the descriptor */shopping.* You can even set this parameter to 1,3. For a web site that

	contains URLs such as http://www.eazykart.com/web/shopping/products.jsp , http://www.eazykart.com/web/shopping/products/travel/bags.jsp , and http://www.eazykart.com/web/shopping/payment.jsp, this specification will result in the following descriptors: */web/products*, */web/products.jsp*, and */web/payment.jsp*.
URL PATTERNS TO BE IGNORED FROM MONITORING	By default, this test does not track requests to the following URL patterns: **.js,*.css,*.jpeg,*.jpg,*.png*. If required, you can remove one/more patterns from this default list, so that such patterns are monitored, or can append more patterns to this list in order to exclude them from monitoring. For instance, to additionally ignore URLs that end with *.gif* and *.bmp* when monitoring, you need to alter the default specification as follows: **.js,*.css,*.jpeg,*.jpg,*.png,*.gif,*.bmp* **Note:** The URL patterns configured here are not just used by the eG agent to filter out unimportant performance data during metrics collection; these patterns are also used by the RUM collector to determine which beacons should be accepted and which ones need to be discarded. Typically, the very first time this test runs and polls the RUM collector for metrics, the eG agent executing this test searches the performance records stored in the collector for data that pertains to the URL patterns configured for exclusion. If such data is found, the agent then ignores that data during metrics collection. This means that during the very first test execution, URL filtering is performed only by the eG agent. During this time, the RUM collector downloads the URL patterns configured against this parameter from the eG agent. Armed with this information, the collector then scans all beacons that browsers send subsequently, and determines if there are any beacons for the excluded URL patterns. If such beacons are found, the collector discards them instantly. Filtering URLs at the collector-level significantly reduces the load on the collector and conserves storage space on the collector; it also minimizes the workload of the eG agent. By additionally filtering URLs at the agent-level, eG makes sure that even if beacons pertaining to excluded URL patterns find their way into the RUM collector, they are captured and siphoned out by the eG agent.
JAVASCRIPT ERRORS TO BE IGNORED	By default, this test alerts administrators to all Javascript errors that occur in the monitored web site/web application. This is why, this parameter is set to *none* by default. Sometimes however, administrators may not want to be notified when certain types of Javascript errors occur – this could be because such errors are harmless or are a normal occurrence in their environment. In such circumstances, you can instruct the eG agent to ignore these errors when monitoring. For this, specify the Java script error message to be ignored in the **JAVASCRIPT ERRORS TO BE IGNORED** text box, in the following format: *<Javascript eror message>:<URL of the page/file where the message originated>*. For instance, say that the *login.html* page in your web site runs a few Java scripts that throw *Object expected* errors, which you want the eG agent to ignore. In this case, your error specification can be as follows: *Object expected:http://www.eazykart.com/web/login.html*. Alternatively, you can provide only

	that text string with which the error message begins in your specification – eg., *Object:http://www.eazykart.com/web/login.html*. Moreover, instead of the complete URL, you can specify just the name of the HTML/jsp/aspx page in which the error is to be ignored – example: *Object:login.html*. Sometimes, the individual web pages in your web site may not run any Java script directly. Instead, these web pages may include links to Java script files that will run the Java script and return the output to the web pages. If you want the eG agent to ignore certain errors thrown by such a Javascript file, then your error pattern specification should include the URL of the Javascript file and not the web page that points to it. This is because, in this case, the file is where the error message originates. For instance, in the same example above, if the *login.html* page points to a *validate.js* file, and you want to ignore the Object expected errors that this JS file throws, your error pattern specification will either be, *Object expected:validate.js*, or *Object:validate.js*. Multiple error message-URL combinations can also be provided as a comma-separated list. The format of your specification will be: *<Javascript error message 1>:<Originating URL 1>,<Javascript error message 2>:<Originating URL 2>,. . .* For example, to ignore the *Object expected* and *Uncaught TypeError* errors in the *login.html* page, use the following specification: *Object:login.html,Uncaught:login.html* Likewise, to ignore the *Object expected* error in the *login.html* page and the *Uncaught TypeError* in the *validate.js* file, your specification will be: *Object:login.html,Uncaught:validate.js* If you want to ignore the *Uncaught TypeError* across all the pages of the web site, your specification will be as follows: Uncaught TypeError:All **Note:** When specifying the *<Javascript error message>* to be ignored, take care of the following: • The error message should not contain any special characters – in particular, the ':' (the colon) and the ',' (the comma) characters should be avoided. • The case of the actual error message and the one specified as part of your error specification should match. This is because, the eG agent performs *case-sensitive* pattern matching.
MAXIMUM HEALTHY TRANSACTIONS IN DD	By default, this parameter is set to 5 indicating that the detailed diagnosis of this test will collect and display metrics related to the top-5 normal/healthy page views, in terms of the *Average page load time*. To identify the top 5, the eG agent sorts all healthy transactions in the ascending order of their *Average page load time*, and picks the first 5

	transactions from this sorted list for display in the detailed diagnosis page and for storing in the eG database. You can however, increase or decrease this value depending upon how many healthy transactions you want to see in your detailed diagnosis, and how well-tuned or well-sized the eG database is. If you do not want the detailed diagnosis to include any healthy transaction, set the value of this parameter to *0*. To view all healthy transactions, set the value of this parameter to *all*. Before setting the value of this parameter to *all*, make sure that you have a well-sized eG database in place, as this setting will store details of every transaction that registers an *Avg page load time* value that is lower than the **SLOW TRANSACTION CUTOFF** specification. On a good day, this can result in numerous transactions, and can consume considerable space in the eG database.
MAXIMUM SLOW TRANSACTIONS IN DD	By default, this parameter is set to *5* indicating that the detailed diagnosis of this test will display metrics related to all the top-5 slow transactions, in terms of the *Average page load time*. To identify the top 5, the eG agent sorts all slow transactions in the descending order of their *Average page load time*, and picks the first 5 transactions from this sorted list for display in the **DETAILED DIAGNOSIS** page and for storing in the eG database. You can however, increase or decrease this value depending upon how many slow transactions you want to see in your detailed diagnosis, and how well-tuned or well-sized the eG database is. If you do not want the detailed diagnosis to include any slow transaction, set the value of this parameter to *0*. To view all slow transactions, set the value of this parameter to *all*. Before setting the value of this parameter to *all*, make sure that you have a well-sized eG database in place, as this setting will store details of every transaction that registers an *Avg page load time* value that is higher than the **SLOW TRANSACTION CUTOFF** specification. On a bad day, this can result in numerous transactions, and can consume considerable space in the eG database.
MAXIMUM ERROR TRANSACTIONS IN DD	By default, this parameter is set to *5*, indicating that the detailed diagnosis of this test will display metrics related to the top-5 transactions that encountered JavaScript errors, based on when those errors occurred. To identify the top 5, the eG agent sorts all error transactions in the descending order of the date/time at which the errors occurred, and picks the first 5 transactions from this sorted list for display in the **DETAILED DIAGNOSIS** page and for storing in the eG database. You can however, increase or decrease this value depending upon how many error transactions you want to see in your detailed diagnosis, and how well-tuned or well-sized the eG database is. If you do not want the detailed diagnosis to include any error transaction, set the value of this parameter to *0*. To view all error transactions, set the value of this parameter to *all*. Before setting the value of this parameter to *all*, make sure that you have a well-sized eG database in place, as this setting will store details of every transaction that

	encounters a JavaScript error. On a bad day, this can result in numerous transactions, and can consume considerable space in the eG database.
SLOW TRANSACTION CUTOFF (MS)	This test reports the count of slow page views and also pinpoints the pages that are slow. To determine whether/not a page is slow, this test uses the **SLOW TRANSACTION CUTOFF** parameter. By default, this parameter is set to *4000 millisecs* (i.e., 4 seconds). This means that, if a page takes more than 4 seconds to load, this test will consider that page as a slow page by default. You can increase or decrease this slow transaction cutoff according to what is 'slow' and what is 'normal' in your environment. **Note:** The default value of this parameter is the same as the default *Maximum threshold* setting of the *Avg page load time* measure – i.e., both are set to *4000 millisecs* by default. While the former helps eG to distinguish between slow and healthy page views for the purpose of providing detailed diagnosis, the latter tells eG when to generate an alarm on *Avg page load time*. For best results, it is recommended that both these settings are configured with the same value at all times. Therefore, if you change the value of one of these configurations, then make sure you update the value of the other as well. For instance, if the **SLOW TRASACTION CUTOFF** is changed to *6000 millisecs*, change the *Maximum Threshold* of the *Avg page load time* measure to *6000* millsecs as well.
SEND ZERO VALUES WHEN THERE IS NO TRAFFIC	By default, this flag is set to **No**. This implies that, if there is no traffic to a monitored web site/web application, then this test will not report any metrics to the eG manager. This also means that, by default, users to the eG monitoring console will not know that there is no traffic to the web site/web application. You can however, ensure that users to the eG monitoring console are informed of the absence of any user activity on the web site/web application. For this, set this flag to **Yes**. If this is done, then the eG agent will report all the metrics of this test to the eG manager, despite the fact that their value is *0*. These zero values will clearly indicate to users that there no traffic to the monitored web site/web application.
PAGELOADTIME CUTOFF (MS)	eG RUM relies on the Navigation API to track page views and capture page load time. Sometimes, the Navigation API can incorrectly report abnormally high values for page load time. To ensure that the eG agent ignores beacons with such page load time values, you can configure in this text box, the maximum value for page load time. By default, this is set to 3600000 (ms). This implies that by default, the eG agent will disregard all beacons that carry page load time values higher than 3600000 milliseconds. You can change this value to suit your environment.
PAGE TYPES TO BE INCLUDED IN DASHBOARD	By default, the eG RUM Dashboard displays details of base page views only. You can optionally include Ajax and/or iFrame views as well in the dashboard, by selecting the relevant options from this list box.
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.
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.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Page views:	Indicates the total number of times pages in this web site/web application were viewed by users using this browser.	Number	This is a good measure of the traffic to your web site/web application from a given browser. A high number of page views from a single browser typically indicates how popular the browser is. Sudden, but significant spikes in the page view count could be a cause for concern, as it could be owing to a malicious virus attack or an unscrupulous attempt to hack your web site/web application. **Note:** An abnormally high value for this measure may not always be a cause for concern; nor would it always indicate a genuine increase in traffic to the web site/web application. If the eG agent monitoring the *Real User Monitor* component is stopped for sometime (say, for maintenance purposes) and then started, or if the eG agent-collector connection breaks and is

Measurement	Description	Measurement Unit	Interpretation
			restored after a while, the eG agent will pull all the metrics that the collector stored locally during the period of its absence, cumulate them, and then display the cumulated values in the eG monitoring console as metrics that pertain to the current measurement period. In reality, these metrics pertain to the entire time period when the eG agent was unavailable. Because of this, the *Page views* measure may indicate a sudden and significant surge in traffic.
Apdex score:	Indicates the apdex score of the web site/web application based on the experience of users using this browser.	Number	**Apdex (Application Performance Index)** is an open standard developed by an alliance of companies. It defines a standard method for reporting and comparing the performance of software applications in computing. Its purpose is to convert measurements into insights about user satisfaction, by specifying a uniform way to analyze and report on the degree to which measured performance meets user expectations. The Apdex method converts many measurements into one number on a uniform scale of 0-to-1 (0 = no users satisfied, 1 = all users satisfied). The resulting Apdex score is a numerical measure of user satisfaction with the performance of enterprise applications. This metric can be used to report on any source of end- user performance measurements for which a performance objective has been defined. The Apdex formulais: $Apdex_t$ = *(Satisfied Count + Tolerating Count / 2) / Total Samples*

Measurement	Description	Measurement Unit	Interpretation
			This is nothing but the number of satisfied samples plus half of the tolerating samples plus none of the frustrated samples, divided by all the samples. A score of 1.0 means all responses were satisfactory. A score of 0.0 means none of the responses were satisfactory. Tolerating responses half satisfy a user. For example, if all responses are tolerating, then the Apdex score would be 0.50. Ideally therefore, the value of this measure should be *1.0*. A value less than *1.0* indicates that the experience of users using this browser to access the target web site/web application has been less than satisfactory.
Average page load time:	Indicates the average time taken by the pages in the web site/web application to load completely on to this browser.	ms	This is the average interval between the time that a user initiates a request and the completion of the page load of the response in the user's browser. In the context of an Ajax request, it ends when the response has been completely processed. By comparing the value of this measure across browsers, you will be able to tell if the page load time is significantly higher for any one browser. If so, then consider it as the first sign of a problem with that browser. Now, use the detailed diagnosis of this measure for the problematic browser to know which specific pages are slow in loading on that browser. Then, check the detailed diagnosis of the other browsers to determine whether/not any slowness has been observed in the same pages when accessed using those browsers. If not, then you can confirm

Measurement	Description	Measurement Unit	Interpretation
			that a problem exists in that one browser only. A browser problem often manifests itself as a delay in DOM download, DOM processing, or page rendering. Therefore, once you zero-in on the problematic browser, check the values of the *Average DOM download time*, *Average DOM processing time*, and *Average page rendering time* measures to ascertain where the browser is losing considerable time – when downloading the HTML document? Or when processing it? Or when rendering the page?
Unique user session:	Indicates the number of distinct users who are currently accessing the web site/web application using this browser.	Number	
Page views per minute:	Indicates the number of times the pages in the web site/web application were viewed per minute using this browser.	Number	This is a good indicator of the level of activity on the web site. An unusually high value for this measure may require investigation.
Normal page view percentage:	Indicates the percentage of page views from this browser that delivered a satisfactory experience to users.	Percent	The value of this measure indicates the percentage of page views in which users have neither experienced any slowness, nor encountered any Javascript errors. Ideally, the value of this measure should be 100%. A value that is slightly less than 100% indicates that the user experience from this browser has not been up to the mark. A value less than 50% is indicative of a serious problem, where most of the page views from this browser are either slow or have encountered Javascript

Measurement	Description	Measurement Unit	Interpretation
			errors. Under such circumstances, to know what exactly is affecting the experience of users using that browser, compare the value of the *Slow page view percentage* with that of the *Javasript error page view percentage* for that browser. This will reveal the reason for the poor user experience – slow pages? or Javascript errors? If slow pages are the problem, use the detailed diagnosis of the *Slow page views* measure of that browser to know which pages are slow and why. If the slowness is indeed owing to the use of an incompatible browser, then the detailed diagnostics will reveal a very high *Average DOM ready time* for each of the slow pages. On the other hand, if the slowness is owing to a network or backend problem, then very high values will be registered for the *Average server connection time* or *Average response available time* measures, as the case may be.
Slow page view percentage:	Indicates the percentage of page views that are slow in loading on this browser.	Percent	Ideally, the value of this measure should be 0. A value over 50% implies that you are in a spot of bother, with over half of the page views being slow. Use the detailed diagnosis of the *Slow page views* measure to identify the slow pages and isolate the root-cause of the slowness – is it the browser? the network? or the backend?
JavaScript error view percentage:	Indicates the percentage of page views from this browser that have encountered JavaScript errors.	Percent	Ideally, the value of this measure should be 0. A value over 50% implies that you are in a spot of bother, with over half of the page views from the browser experiencing JavaScript errors.

Measurement	Description	Measurement Unit	Interpretation
Slow page views (Tolerating & Frustrated):	Indicates the number of times pages in the target web site/web application took very long to be viewed on this browser.	Number	A page view is considered to be slow when the average time taken to load that page exceeds the **SLOW TRANSACTION CUTOFF** configured for this test. Ideally, a page should load quickly. The value 0 is hence desired for this measure. If the value of this measure is high, it indicates that users using this browser frequently experienced slowness when accessing pages in the web site/web application. To know which page views are slow and why, use the detailed diagnosis of this measure.
JavaScript error page views:	Indicates the number of times JavaScript errors occurred when viewing the pages in the target web site/web application using this browser.	Number	Ideally, the value of this measure should be 0. A high value indicates that many JavaScript errors are occurring when accessing the web site/web application using this browser. Use the detailed diagnosis of this measure to identify the error pages and to know what Javascript error has occurred in which page. This will greatly aid troubleshooting!
Satisfied page views:	Indicates the number of times pages were viewed from this browser without any slowness.	Number	A page view is considered to be slow when the average time taken to load that page exceeds the **SLOW TRANSACTION CUTOFF** configured for this test. If this **SLOW TRANSACTION CUTOFF** is not exceeded, then the page view is deemed to be 'satisfactory'. To know which page views are satisfactory, use the detailed diagnosis of this measure. Ideally, the value of this measure should be the same as that of the *Page views* measure. If not, then it indicates that one/more page views are slow – i.e., have violated the **SLOW TRANSACTION CUTOFF**.

Measurement	Description	Measurement Unit	Interpretation
			If the value of this measure is much lesser than the value of the *Tolerating page views* and the *Frustrated page views*, it is a clear indicator that the experience of users using this browser is below-par. In such a case, use the detailed diagnosis of the *Tolerating page views* and *Frustrated page views* measures to know which pages are slow and why.
Tolerating page views:	Indicates the number of tolerating page views from this browser.	Number	If the *Average page load time* of a page exceeds the **SLOW TRANSACTION CUTTOFF** configuration of this test, but is less than 4 times the **SLOW TRANSACTION CUTOFF** (i.e., < 4 * **SLOW TRANSACTION CUTOFF**), then such a page view is considered to be a **Tolerating page view**. Ideally, the value of this measure should be 0. A value higher than that of the *Satisfied page views* measure is a cause for concern, as it implies that the overall user experience from this browser is less than satisfactory. To know which pages are contributing to this sub-par experience, use the detailed diagnosis of this measure. The detailed metrics will also enable you to accurately isolate what is causing the tolerating page views – a problem with the browser? network? or backend?
Frustrated page views:	Indicates the number of frustrated page views from this browser.	Number	If the *Average page load time* of a page is over 4 times the **SLOW TRANSACTION CUTTOFF** configuration of this test(i.e., > 4 * **SLOW TRANSACTION CUTOFF**), then such a page view is considered to be a **Frustrated page view**.

Measurement	Description	Measurement Unit	Interpretation
			Ideally, the value of this measure should be 0. A value higher than that of the *Satisfied page views* measure is a cause for concern, as it implies that the experience of users using this browser has been less than satisfactory. To know which pages are contributing to this sub-par experience, use the detailed diagnosis of this measure. The detailed metrics will also enable you to accurately isolate what is causing the frustrated page views – a problem with the browser? network? or backend?
Desktop page views:	Indicates the number of times pages in the target web site/web application were accessed by desktop users using this browser.	Number	To know which pages were accessed from client desktops using this browser and to evaluate the experience of the desktop users with each of these pages, use the detailed diagnosis of this measure. In the process, slow pages can be identified and the reason for the slowness can be pinpointed.
Mobile page views:	Indicates the number of times pages in the target web site/web application were accessed by mobile phone users using this browser.	Number	To know which pages in the web site were accessed from mobile phones using this browser and to evaluate the experience of the mobile phone users with each of the pages, use the detailed diagnosis of this measure. In the process, slow pages can be identified and the reason for the slowness can be pinpointed.
Tablet page views:	Indicates the number of times pages in the target web site/web application were accessed from tablets using this browser.	Number	To know which pages in the web site were accessed from tablets using this browser and to evaluate the experience of the tablet users with each of the pages, use the detailed diagnosis of this measure. In the process, slow pages can be identified and the reason for the

Measurement	Description	Measurement Unit	Interpretation
			slowness can be pinpointed.
Average front end time:	Indicates the interval between the arrival of the first byte of text response and the completion of the response page rendering by this browser.	ms	In the Figure 3.76 that depicts a typical page loading process, the *Average front end time* denotes the time from the *responseStart* event to the *loadEventEnd*. This process includes document downloading, processing, and page rendering. This time is therefore the sum of the *Average DOM ready time* and the *Average page rendering time*. If the *Average page load time* of the web site/web application exceeds its threshold, then you may want to compare the value of this measure with that of the *Average server connection time* and *Average response available time* to zoom into the source of the slowness – is it the browser? the network? or the backend? If the *Average front end time* is the highest, it indicates that the problem is with browser – this can be attributed to a slowdown in page rendering or in DOM building. To nail the precise cause of the slowdown, compare the values of the *Average DOM processing time*, *Average DOM download time*, and the *Average page rendering time* measures. On the basis of this comparison, you will be able to tell whether the slowness occurred when downloading the document, when processing it, or when rendering the response page in the browser.
Average page rendering time:	Indicates the time taken to complete the download of remaining resources, including images, and to finish rendering the page in	ms	A high value of this measure indicates that this browser is taking too long to render the page. This can adversely impact the *Average front end time*, which in turn can prolong the *Average page load*

Measurement	Description	Measurement Unit	Interpretation
	this browser.		*time*. Ideally therefore, the value of this measure should be low.
Average DOM ready time:	Indicates the time taken by this browser to make the complete HTML document (DOM) available for JavaScript to apply rendering logic.	ms	The value of this measure is the sum of the *Average DOM download time* and the *Average DOM processing time* measures. If the value of this measure is very high, then you may want to compare the *Average DOM download time* and the *Average DOM processing time* measures to figure out what is delaying DOM building – downloading? Or processing? A high value for this measure can adversely impact the *Average front end time* , which in turn can prolong the *Average page load time*. Ideally therefore, the value of this measure should be low.
Average DOM download time:	Indicates the time taken to download the complete HTML document on this browser.	ms	Higher the download time of the document, longer will be the time taken to make the document available for page rendering. As a result, the overall user experience will be affected when this browser is used! This is why, a low value is desired for this measure at all times.
Average DOM processing time:	Indicates the time taken by this browser to build the Document Object Model (DOM) and make it available for JavaScript to apply rendering logic.	ms	An unusually high value for this measure is a clear indicator that DOM building is taking longer than normal. In consequence, page rendering will be delayed, thus adversely impacting user experience when this browser is used. Ideally therefore, the value of this measure should be low.
Average first byte time:	Indicates the interval between the time that a user initiates a request and the time that this browser	ms	Going by Figure 3.76, the *Average first byte time* is the time that elapsed between *navigationStart* and *responseStart*. The value of this measure

Measurement	Description	Measurement Unit	Interpretation
	receives the first response byte. In the context of an Ajax request, this is the interval between the Ajax request dispatch and the time that the browser receives the first response byte.		is also the sum of *Average response available time* , *Average DNS lookup time*, and *Average TCP connection time*. This means that an abnormal increase in any of the above-mentioned time values will increase the value of this measure. If the first response byte from the target web site/web application is itself received slowly, it is bound to have a cascading effect on all events that follow – such as, document downloading, processing, and page rendering. Ultimately, this will impact the page load time as seen by end-users. This is why, if the *Average first byte time* violates its threshold, administrators need to instantly switch to the troubleshooting mode and rapidly isolate what is causing it – is DNS lookup taking a long time? is the network connection to the web site/web application latent? or is the web server/web application server hosting the web site slow in processing requests? By comparing the values of the *Average response available time*, *Average DNS lookup time* , and *Average TCP connection time* measures, administrators can swiftly and accurately figure out the exact reason why there was a delay in receiving the first response byte. If this comparison reveals that the *Average DNS lookup time* is the highest, it implies that domain name resolution by the DNS server is taking a long time and impacting responsiveness. If the *Average TCP connection time* is found to be the culprit, then blame the network connection for delaying the transmission

Measurement	Description	Measurement Unit	Interpretation
			of the response byte. If the *Average reponse available time* is higher than the rest, you can be rest assured that the source of the problem lies with the server hosting the web site/web application.
Average response available time:	Indicates the interval between the start of processing of a request on this browser to when this browser receives the response.	ms	In Figure 3.76, the *Average response available* time is the time spent between the *requestStart* event and *responseStart* event. Ideally, a low value is desired for this measure, as high values will certainly hurt the *Apdex scrore* of the web site/web application. The key factor that can influence the value of this measure is the request processing ability of the web server/web application server that is hosting the web site/web application being monitored. Any slowdown in the backend web server/web application server – caused by the lack of adequate processing power in or improper configuration of the backend server – can significantly delay request processing by the server. In its aftermath, the *Average response available time* will increase, leaving users with an unsatisfactory experience with the web site/web application.
Average server connection time:	Indicates the elapsed time since a user using this browser initiates a request and the start of fetching the response document from the server or application.	ms	In Figure 3.14, the time spent between *navigationStart* and *requestStart* makes up the *Average server connection time*. This includes the time to perform DNS lookups and the time to establish a TCP connection with the server. In other words, the value of this measure is nothing but the sum of the *Average DNS lookup time* and the *Average TCP*

Measurement	Description	Measurement Unit	Interpretation
			connection time measures. Ideally, the value of this measure should be low. A very high value will often end up delaying page loading and degrading the quality of the web site service. In the event that the server connection time is high therefore, simply compare the values of the *Average DNS lookup time* and *Average TCP connection time* measures to know to what this delay can be attributed – a delay in domain name resolution? Or a poor network connection to the server?
Average DNS lookup time:	Indicates the time taken by this browser to perform the domain lookup for connecting to the web site/web application.	ms	A high value for this measure will not only affect DNS lookup, but will also impact the *Average server connection time* and *Average page load time* of the web site/web application. This naturally will have a disastrous effect on user experience.
Average TCP connection time:	Indicates the time taken by this browser to establish a TCP connection with the server.	ms	A bad network connection between the browser client and the server can delay TCP connections to the server As a result, the *Average server connection time* too will increase, thus impacting page load time and overall user experience with the web site/web application.

3.6.3 Devices Test

Users today do not rely on their PCs alone to access web applications or consume web-based services. Users now work with a wide range of devices - from iPhones to iPads to Tablets - that allow quick, easy, and on-the-go access to the web. On any typical day therefore, a web site/web application may be bombarded with requests from a gamut of devices. Since the technology driving each of these devices is very distinct, there is a high likelihood that the experience of users may vary with the device. Insights into the user experience per device type can help administrators figure out whether the slowness experienced or errors

encountered by users are owing to the device they are using, or due to other factors such as the network or the backend. The **Devices** test helps administrators with this! This test automatically discovers the types of devices used to launch the target web site/web application and reports the average page load time per device type. The percentage of JavaScript errors encountered by the users of each device type is also reported. This way, the test rapidly pinpoints those device types, the users of which are experiencing problems when accessing the web site/web application.

For every device type, the test also provides a breakup of the page load time, which clearly indicates where the bottleneck lies - is it with the device? the network? or the backend? Detailed diagnostics provided by the test lead administrators to the specific web pages that are slow. Error pages and the errors affecting them are also offered as part of the detailed diagnosis. Based on these inputs, administrators can either attempt to tweak the application to support the 'unfriendly' devices or add that device type to their list of incompatible devices.

Target of the test : A web site/web application managed as a *Real User Monitor*

Agent deploying the test : A remote agent

Outputs of the test : One set of results for each browser used for accessing the web site/web application being monitored.

Test parameters:

TEST PERIOD	How often should the test be executed
PROXYHOST	If the eG agent communicates with the RUM collector via a proxy, then specify the IP address/fully-qualified host name of the proxy server here. By default, this is set to *none*, indicating that the eG agent does not communicate with the collector via a proxy.
PROXYPORT	If the eG agent communicates with the RUM collector via a proxy, then specify the port at which the proxy server listens for requests from the eG agent. By default, this is set to *none*, indicating that the eG agent does not communicate with the collector via a proxy.
PROXYUSERNAME and **PROXYPASSWORD**	If the eG agent communicates with the RUM collector via a proxy server, and if proxy server requires authentication, then specify valid credentials for authentication against **PROXYUSERNAME** and **PROXYPASSWORD**. If no proxy server is used or if the proxy server used does not require authentication, then set the **PROXYUSERNAME** and **PROXYPASSWORD** to *none*.
CONFIRM PASSWORD	Confirm the **PROXYPASSWORD** by retyping it here. **Note:** If you **Reconfigure** the test later to change the values of the **PROXYUSERNAME**, **PROXYPASSWORD**, and **CONFIRM PASSWORD** parameters, then such changes will be effected only if the eG remote agent monitoring the Real User Monitor component is restarted.
DO YOU WANT TO LIMIT THE PAGE VIEWS?	By default, the eG RUM monitors all requests to a managed web site. This is why, this flag is set to **No** by default. However, in case of web sites that receive thousands of hits every day, monitoring each page view may add significantly to the overhead of the eG agent and may also increase the size of the eG database considerably. To reduce the

	strain on both the eG agent and the eG backend, you may want to restrict the monitoring scope of this test to a few page visits. To achieve this, first set this flag to **Yes**. This will invoke the option depicted by the below figure.  Figure 3.79: Configuring the number of allowed page visits By default, the **MAXIMUM ALLOWED PAGE VISITS PER DAY** is set to *100000*. This implies that the test will consider only the first 100000 requests in a day for monitoring. All page visits beyond 100000 will by default be excluded from the test's monitoring purview. You can increase or decrease this limit, if you so need.
URL SEGMENTS TO BE USED AS GROUPED URL	**This parameter is applicable to the Page Groups test alone**. The **Page Groups** test groups URLs based on the URL segments configured for monitoring and reports aggregated response time metrics for every group. Using this parameter, you can specify a comma-separated list of URL segment numbers based on which the pages are to be grouped. URL segments are the parts of a URL (after the base URL) or path delimited by slashes. So if you had the URL: http://www.eazykart.com/web/shopping/login.jsp, then http://www.eazykart.com will be the base URL or domain, */web* will be the first URL segment, */shopping* will be the second URL segment, and */login.jsp* will be the third URL segment. By default, this parameter is set to *1,2*. This default setting, when applied to the sample URL provided above, implies that the eG agent will aggregate request and response time metrics to all instrumented web pages (i.e., web pages with the code snippet) under the URL */web/shopping*. Here, */web* corresponds to the specification 1 (URL segment 1) and */shopping* corresponds to the specification 2 (URL segment 2) in the default value *1,2*. This in turn means that, if the web site consists of pages such as http://www.eazykart.com/web/shopping/products.jsp , http://www.eazykart.com/web/shopping/products/travel/bags.jsp , http://www.eazykart.com/web/shopping/payment.jsp, etc., then the eG agent will track the requests to and responses from all these web pages, aggregate the results, and present the aggregated metrics for the descriptor */web/shopping*. This way, the test will create different page groups based on each of the second-level URL segments in the managed web site – eg., */web/movies* , */web/travel* , */web/reservations* , */partner/contacts* etc. – and will report aggregated metrics for each group so created. If you want, you can override the default setting by providing different URL segment numbers here. For instance, your specification can be just *2*. In this case, for the URL http://www.eazykart.com/web/shopping/login.jsp, the test will report metrics for the descriptor */shopping.* You can even set this parameter to *1,3.* For a web site that contains URLs such as http://www.eazykart.com/web/shopping/products.jsp , http://www.eazykart.com/web/shopping/products/travel/bags.jsp , and http://www.eazykart.com/web/shopping/payment.jsp, this specification will result in the following descriptors: */web/products*, */web/products.jsp*, and */web/payment.jsp*.

URL PATTERNS TO BE IGNORED FROM MONITORING	By default, this test does not track requests to the following URL patterns: **.js,*.css,*.jpeg,*.jpg,*.png*. If required, you can remove one/more patterns from this default list, so that such patterns are monitored, or can append more patterns to this list in order to exclude them from monitoring. For instance, to additionally ignore URLs that end with *.gif* and *.bmp* when monitoring, you need to alter the default specification as follows: **.js,*.css,*.jpeg,*.jpg,*.png,*.gif,*.b* **Note:** The URL patterns configured here are not just used by the eG agent to filter out unimportant performance data during metrics collection; these patterns are also used by the RUM collector to determine which beacons should be accepted and which ones need to be discarded. Typically, the very first time this test runs and polls the RUM collector for metrics, the eG agent executing this test searches the performance records stored in the collector for data that pertains to the URL patterns configured for exclusion. If such data is found, the agent then ignores that data during metrics collection. This means that during the very first test execution, URL filtering is performed only by the eG agent. During this time, the RUM collector downloads the URL patterns configured against this parameter from the eG agent. Armed with this information, the collector then scans all beacons that browsers send subsequently, and determines if there are any beacons for the excluded URL patterns. If such beacons are found, the collector discards them instantly. Filtering URLs at the collector-level significantly reduces the load on the collector and conserves storage space on the collector; it also minimizes the workload of the eG agent. By additionally filtering URLs at the agent-level, eG makes sure that even if beacons pertaining to excluded URL patterns find their way into the RUM collector, they are captured and siphoned out by the eG agent.
JAVASCRIPT ERRORS TO BE IGNORED	By default, this test alerts administrators to all Javascript errors that occur in the monitored web site/web application. This is why, this parameter is set to *none* by default. Sometimes however, administrators may not want to be notified when certain types of Javascript errors occur – this could be because such errors are harmless or are a normal occurrence in their environment. In such circumstances, you can instruct the eG agent to ignore these errors when monitoring. For this, specify the Java script error message to be ignored in the **JAVASCRIPT ERRORS TO BE IGNORED** text box, in the following format: *<Javascript eror message>:<URL of the page/file where the message originated>*. For instance, say that the *login.html* page in your web site runs a few Java scripts that throw *Object expected* errors, which you want the eG agent to ignore. In this case, your error specification can be as follows: *Object expected:http://www.eazykart.com/web/login.html*. Alternatively, you can provide only that text string with which the error message begins in your specification – eg., *Object:http://www.eazykart.com/web/login.html*. Moreover, instead of the complete URL, you can specify just the name of the HTML/jsp/aspx page in which the error is to be ignored – example: *Object:login.html*.

	Sometimes, the individual web pages in your web site may not run any Java script directly. Instead, these web pages may include links to Java script files that will run the Java script and return the output to the web pages. If you want the eG agent to ignore certain errors thrown by such a Javascript file, then your error pattern specification should include the URL of the Javascript file and not the web page that points to it. This is because, in this case, the file is where the error message originates. For instance, in the same example above, if the *login.html* page points to a *validate.js* file, and you want to ignore the *Object expected errors* that this JS file throws, your error pattern specification will either be, *Object expected:validate.js*, or *Object:validate.js*. Multiple error message-URL combinations can also be provided as a comma-separated list. The format of your specification will be: <Javascript error message 1>:<Originating URL 1>,<Javascript error message 2>:<Originating URL 2>,. . . For example, to ignore the *Object expected* and *Uncaught TypeError* errors in the *login.html* page, use the following specification: *Object:login.html,Uncaught:login.html* Likewise, to ignore the *Object expected* error in the *login.html* page and the *Uncaught TypeError* in the *validate.js* file, your specification will be: Object:login.html,Uncaught:validate.js If you want to ignore the *Uncaught TypeError* across all the pages of the web site, your specification will be as follows: *Uncaught TypeError:All* **Note:** When specifying the *<Javascript error message>* to be ignored, take care of the following: • The error message should not contain any special characters – in particular, the ':' (the colon) and the ',' (the comma) characters should be avoided. • The case of the actual error message and the one specified as part of your error specification should match. This is because, the eG agent performs *case-sensitive* pattern matching.
MAXIMUM HEALTHY TRANSACTIONS IN DD	By default, this parameter is set to *5* indicating that the detailed diagnosis of this test will collect and display metrics related to the top-5 normal/healthy page views, in terms of the *Average page load time*. To identify the top 5, the eG agent sorts all healthy transactions in the ascending order of their *Average page load time*, and picks the first 5 transactions from this sorted list for display in the **DETAILED DIAGNOSIS** page and for storing in the eG database. You can however, increase or decrease this value depending upon how many healthy transactions you want to see in your detailed diagnosis, and how well-tuned or well-

	sized the eG database is. If you do not want the detailed diagnosis to include any healthy transaction, set the value of this parameter to *0*. To view all healthy transactions, set the value of this parameter to *all*. Before setting the value of this parameter to *all*, make sure that you have a well-sized eG database in place, as this setting will store details of every transaction that registers an *Avg page load time* value that is lower than the **SLOW TRANSACTION CUTOFF** specification. On a good day, this can result in numerous transactions, and can consume considerable space in the eG database.
MAXIMUM SLOW TRANSACTIONS IN DD	By default, this parameter is set to *5* indicating that the detailed diagnosis of this test will display metrics related to all the top-5 slow transactions, in terms of the *Average page load time*. To identify the top 5, the eG agent sorts all slow transactions in the descending order of their *Average page load time*, and picks the first 5 transactions from this sorted list for display in the **DETAILED DIAGNOSIS** page and for storing in the eG database. You can however, increase or decrease this value depending upon how many slow transactions you want to see in your detailed diagnosis, and how well-tuned or well-sized the eG database is. If you do not want the detailed diagnosis to include any slow transaction, set the value of this parameter to *0*. To view all slow transactions, set the value of this parameter to *all*. Before setting the value of this parameter to *all*, make sure that you have a well-sized eG database in place, as this setting will store details of every transaction that registers an *Avg page load time* value that is higher than the **SLOW TRANSACTION CUTOFF** specification. On a bad day, this can result in numerous transactions, and can consume considerable space in the eG database.
MAXIMUM ERROR TRANSACTIONS IN DD	By default, this parameter is set to *5*, indicating that the detailed diagnosis of this test will display metrics related to the top-5 transactions that encountered JavaScript errors, based on when those errors occurred. To identify the top 5, the eG agent sorts all error transactions in the descending order of the date/time at which the errors occurred, and picks the first 5 transactions from this sorted list for display in the **DETAILED DIAGNOSIS** page and for storing in the eG database. You can however, increase or decrease this value depending upon how many error transactions you want to see in your detailed diagnosis, and how well-tuned or well-sized the eG database is. If you do not want the detailed diagnosis to include any error transaction, set the value of this parameter to *0*. To view all error transactions, set the value of this parameter to *all*. Before setting the value of this parameter to *all*, make sure that you have a well-sized eG database in place, as this setting will store details of every transaction that encounters a JavaScript error. On a bad day, this can result in numerous transactions, and can consume considerable space in the eG database.
SLOW TRANSACTION	This test reports the count of slow page views and also pinpoints the pages that are slow. To determine whether/not a page is slow, this test uses the **SLOW TRANSACTION**

CUTOFF (MS)	**CUTOFF** parameter. By default, this parameter is set to *4000 millisecs* (i.e., 4 seconds). This means that, if a page takes more than 4 seconds to load, this test will consider that page as a slow page by default. You can increase or decrease this slow transaction cutoff according to what is 'slow' and what is 'normal' in your environment. **Note:** The default value of this parameter is the same as the default *Maximum threshold* setting of the *Avg page load time* measure – i.e., both are set to *4000 millisecs* by default. While the former helps eG to distinguish between slow and healthy page views for the purpose of providing detailed diagnosis, the latter tells eG when to generate an alarm on *Avg page load time*. For best results, it is recommended that both these settings are configured with the same value at all times. Therefore, if you change the value of one of these configurations, then make sure you update the value of the other as well. For instance, if the **SLOW TRASACTION CUTOFF** is changed to *6000 millisecs*, change the *Maximum Threshold* of the *Avg page load time* measure to *6000* millsecs as well.
SEND ZERO VALUES WHEN THERE IS NO TRAFFIC	By default, this flag is set to **No**. This implies that, if there is no traffic to a monitored web site/web application – i.e., if all measures of this test return only the value *0* - then the eG agent will not report these metrics to the eG manager. This also means that, by default, users to the eG monitoring console will not know that there is no traffic to the web site/web application. You can however, ensure that users to the eG monitoring console are informed of the absence of any user activity on the web site/web application. For this, set this flag to **Yes**. If this is done, then the eG agent will report all the metrics of this test to the eG manager, despite the fact that their value is *0*. These zero values will clearly indicate to users that there no traffic to the monitored web site/web application.
PAGELOADTIME CUTOFF (MS)	eG RUM relies on the Navigation API to track page views and capture page load time. Sometimes, the Navigation API can incorrectly report abnormally high values for page load time. To ensure that the eG agent ignores beacons with such page load time values, you can configure in this text box, the maximum value for page load time. By default, this is set to 3600000 (ms). This implies that by default, the eG agent will disregard all beacons that carry page load time values higher than 3600000 milliseconds. You can change this value to suit your environment.
PAGE TYPES TO BE INCLUDED IN DASHBOARD	By default, the eG RUM Dashboard displays details of base page views only. You can optionally include Ajax and/or iFrame views as well in the dashboard, by selecting the relevant options from this list box.
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.
DETAILED	To make diagnosis more efficient and accurate, the eG Enterprise suite embeds an

DIAGNOSIS	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.

Measurements made by the test

Measurements	Description	Measurement Unit	Interpretation
Page views:	Indicates the total number of times pages in the web site/web application were viewed by users using this device type.	Number	This is a good measure of the traffic to your web site/web application from a given device type. A high number of page views from a single device type typically indicates how popular the device is. Sudden, but significant spikes in the page view count could be a cause for concern, as it could be owing to a malicious virus attack or an unscrupulous attempt to hack your web site/web application. **Note:** An abnormally high value for this measure may not always be a cause for concern; nor would it always indicate a genuine increase in traffic to the web site/web application. If the eG agent monitoring the Real User Monitor component is stopped for sometime (say, for maintenance purposes) and then started, or if the eG agent-collector connection breaks and is restored after a while, the eG agent will pull all the metrics that the collector stored locally during the period of its absence, cumulate them, and then

Measurements	Description	Measurement Unit	Interpretation
			display the cumulated values in the eG monitoring console as metrics that pertain to the current measurement period. In reality, these metrics pertain to the entire time period when the eG agent was unavailable. Because of this, the Page views measure may indicate a sudden and significant surge in traffic.
Apdex score:	Indicates the apdex score of the web site/web application based on the experience of users using this device type.	Number	**Apdex** (Application Performance Index) is an open standard developed by an alliance of companies. It defines a standard method for reporting and comparing the performance of software applications in computing. Its purpose is to convert measurements into insights about user satisfaction, by specifying a uniform way to analyze and report on the degree to which measured performance

Measurements	Description	Measurement Unit	Interpretation
			meets user expectations. The Apdex method converts many measurements into one number on a uniform scale of 0-to-1 (0 = no users satisfied, 1 = all users satisfied). The resulting Apdex score is a numerical measure of user satisfaction with the performance of enterprise applications. This metric can be used to report on any source of end- user performance measurements for which a performance objective has been defined. The Apdex formulais: Apdext = (Satisfied Count + Tolerating Count / 2) / Total Samples This is nothing but the number of satisfied samples plus half of the tolerating samples plus none of the frustrated samples, divided by all the samples. A score of 1.0 means all responses were satisfactory. A score of 0.0 means none of the responses were satisfactory. Tolerating responses half satisfy a user. For example, if all responses are tolerating, then the Apdex score would be 0.50. Ideally therefore, the value of this measure should be 1.0. A value less than 1.0 indicates that the experience of users using this type of device to access the target web site/web application has been less than satisfactory.
Average page load time:	Indicates the average time taken by the pages in the web site/web application to load completely on the	ms	This is the average interval between the time that a user initiates a request and the completion of the page load of the response in the user's browser on this

Measurements	Description	Measurement Unit	Interpretation
	browser launched from this device.		device. In the context of an Ajax request, it ends when the response has been completely processed. By comparing the value of this measure across device types, you will be able to tell if the page load time is significantly higher for any one type of device. If so, then consider it as the first sign of a problem with that device type. Now, use the detailed diagnosis of this measure for the problematic device to know which specific pages are slow in loading on that device. Then, check the detailed diagnosis of the other devices to determine whether/not any slowness has been observed in the same pages when accessed using those devices. If not, then you can confirm that a problem exists in that one device type only. A device problem often manifests itself as a delay in DOM download, DOM processing, or page rendering. Therefore, once you zero-in on the problematic device, check the values of the Average DOM download time, Average DOM processing time, and Average page rendering time measures to ascertain where the device is losing considerable time – when downloading the HTML document? when processing it? or when rendering the page?
Unique user session:	Indicates the number of distinct users who are currently accessing the web site/web application using this device type.	Number	
Page views per	Indicates the number of	Number	This is a good indicator of the level of

Measurements	Description	Measurement Unit	Interpretation
minute:	times the pages in the web site/web application were viewed per minute using this device type.		activity on the web site. An unusually high value for this measure may require investigation.
Normal page view percentage:	Indicates the percentage of page views from this device type that delivered a satisfactory experience to users.	Percent	The value of this measure indicates the percentage of page views in which users have neither experienced any slowness, nor encountered any Javascript errors. Ideally, the value of this measure should be 100%. A value that is slightly less than 100% indicates that the user experience from this device has not been up to the mark. A value less than 50% is indicative of a serious problem, where most of the page views from this device are either slow or have encountered Javascript errors. Under such circumstances, to know what exactly is affecting the experience of users using that device, compare the value of the Slow page view percentage with that of the Javasript error page view percentage for that browser. This will reveal the reason for the poor user experience – slow pages? or Javascript errors? If slow pages are the problem, use the detailed diagnosis of the Slow page views measure of that device to know which pages are slow and why. If the slowness is indeed owing to the use of an incompatible device, then the detailed diagnostics will reveal a very high Average DOM ready time or Average page rendering time for each of the slow pages. On the other hand, if the slowness is owing to a network or backend problem, then very high values will be registered for the Average server

Measurements	Description	Measurement Unit	Interpretation
			connection time or Average response available time measures, as the case may be.
Slow page view percentage:	Indicates the percentage of page views that are slow in loading on this device type.	Percent	Ideally, the value of this measure should be 0. A value over 50% implies that you are in a spot of bother, with over half of the page views being slow. Use the detailed diagnosis of the Slow page views measure to identify the slow pages and isolate the root- cause of the slowness – is it the device? the network? or the backend?
JavaScript error view percentage:	Indicates the percentage of page views from this device that have encountered JavaScript errors.	Percent	Ideally, the value of this measure should be 0. A value over 50% implies that you are in a spot of bother, with over half of the page views from the device experiencing JavaScript errors.
Slow page views (Tolerating & Frustrated):	Indicates the number of page views from this device that were slow.	Number	A page view is considered to be slow when the average time taken to load that page exceeds the slow transaction cutoff configured for this test. Ideally, a page should load quickly. The value 0 is hence desired for this measure. If the value of this measure is high, it indicates that users of this device frequently experienced slowness when accessing pages in the web site/web application. To know which page views are slow and why, use the detailed diagnosis of this measure.
JavaScript error page views:	Indicates the number of times JavaScript errors occurred when accessing pages from this device.	Number	Ideally, the value of this measure should be 0. A high value indicates that many JavaScript errors are occurring when viewing pages from this device. Use the detailed diagnosis of this measure to

Measurements	Description	Measurement Unit	Interpretation
			identify the error pages and to know what Javascript error has occurred in which page. This will greatly aid troubleshooting!
Satisfied page views:	Indicates the number of times pages were viewed from this device without any slowness.	Number	A page view is considered to be slow when the average time taken to load that page exceeds the slow transaction cutoff configured for this test. If this slow transaction cutoff is not exceeded, then the page view is deemed to be 'satisfactory'. To know which page views are satisfactory, use the detailed diagnosis of this measure. Ideally, the value of this measure should be the same as that of the Page views measure. If not, then it indicates that one/more page views are slow – i.e., have violated the slow transaction cutoff. If the value of this measure is much lesser than the value of the Tolerating page views and the Frustrated page views, it is a clear indicator that the experience of users using this device is below-par. In such a case, use the detailed diagnosis of the Tolerating page views and Frustrated page views measures to know which pages are slow and why.
Tolerating page views:	Indicates the number of tolerating page views from this device type.	Number	If the *Average page load time* of a page exceeds the slow transaction cuttoff configuration of this test, but is less than 4 times the slow transaction cutoff (i.e., < 4 * slow transaction cutoff), then such a page view is considered to be a Tolerating page view. Ideally, the value of this measure should

Measurements	Description	Measurement Unit	Interpretation
			be 0. A value higher than that of the Satisfied page views measure is a cause for concern, as it implies that the overall user experience from this device is less than satisfactory. To know which pages are contributing to this sub- par experience, use the detailed diagnosis of this measure. The detailed metrics will also enable you to accurately isolate what is causing the tolerating page views - a problem with the device? network? or backend?
Frustrated page views:	Indicates the number of frustrated page views from this device type.	Number	If the *Average page load time* of a page is over 4 times the slow transaction cuttoff configuration of this test(i.e., > 4 * slow transaction cutoff), then such a page view is considered to be a Frustrated page view. Ideally, the value of this measure should be 0. A value higher than that of the Satisfied page views measure is a cause for concern, as it implies that the experience of users using this device has been less than satisfactory. To know which pages are contributing to this sub-par experience, use the detailed diagnosis of this measure. The detailed metrics will also enable you to accurately isolate what is causing the frustrated page views – a problem with the device? network? or backend?
Average front end time:	Indicates the interval between the arrival of the first byte of text response and the completion of the response page rendering by the browser on this device type.	ms	In the Figure 3.76 that depicts a typical page loading process, the Average front end time denotes the time from the responseStart event to the loadEventEnd. This process includes document downloading, processing, and page rendering. This time is therefore the

Measurements	Description	Measurement Unit	Interpretation
			sum of the Average DOM ready time and the Average page rendering time. If the *Average page load time* exceeds its threshold, then you may want to compare the value of this measure with that of the Average server connection time and Average response available time to zoom into the source of the slowness – is it the device? the network? or the backend? If the *Average front end time* is the highest, it indicates that the problem is with device - this can be attributed to a slowdown in page rendering or in DOM building. To nail the precise cause of the slowdown, compare the values of the *Average DOM processing time*, *Average DOM download time*, and the *Average page rendering time* measures. On the basis of this comparison, you will be able to tell whether the slowness occurred when downloading the document, when processing it, or when rendering the response page in the browser on this device.
Average page rendering time:	Indicates the time taken to complete the download of remaining resources, including images, and to finish rendering the page in the browser on this device type.	ms	A high value of this measure indicates that this device is taking too long to render the page. This can adversely impact the Average front end time, which in turn can prolong the Average page load time. Ideally therefore, the value of this measure should be low.
Average DOM ready time:	Indicates the time taken by the browser on this device type to make the complete HTML document (DOM) available for JavaScript to apply rendering logic.	ms	The value of this measure is the sum of the *Average DOM download time* and the *Average DOM processing time* measures. If the value of this measure is very high, then you may want to compare the *Average DOM download time* and the

Measurements	Description	Measurement Unit	Interpretation
			Average DOM processing time measures to figure out what is delaying DOM building - downloading? Or processing? A high value for this measure can adversely impact the Average front end time, which in turn can prolong the Average page load time. Ideally therefore, the value of this measure should be low.
Average DOM download time:	Indicates the time taken to download the complete HTML document on the browser on this device type.	ms	Higher the download time of the document, longer will be the time taken to make the document available for page rendering. As a result, the overall user experience will be affected when the device type is used! This is why, a low value is desired for this measure at all times.
Average DOM processing time:	Indicates the time taken by the browser on this device type to build the Document Object Model (DOM) and make it available for JavaScript to apply rendering logic.	ms	An unusually high value for this measure is a clear indicator that DOM building is taking longer than normal. In consequence, page rendering will be delayed, thus adversely impacting user experience when this device is used. Ideally therefore, the value of this measure should be low.
Average first byte time:	Indicates the interval between the time that a user initiates a request and the time that the browser on thisdevice type receives the first response byte. In the context of an Ajax request, this is the interval between the Ajax request dispatch and the time that the browser on this device type receives the first response byte.	ms	Going by Figure 3.76, the Average first byte time is the time that elapsed between navigationStart and responseStart. The value of this measure is also the sum of *Average response available time* , *Average DNS lookup time*, and *Average TCP connection time*. This means that an abnormal increase in any of the above-mentioned time values will increase the value of this measure. If the first response byte from the target

Measurements	Description	Measurement Unit	Interpretation
			web site/web application is itself received slowly, it is bound to have a cascading effect on all events that follow - such as, document downloading, processing, and page rendering. Ultimately, this will impact the page load time as seen by end-users. This is why, if the Average first byte time violates its threshold, administrators need to instantly switch to the troubleshooting mode and rapidly isolate what is causing it - is DNS lookup taking a long time? is the network connection to the web site/web application latent? or is the web server/web application server hosting the web site slow in processing requests? By comparing the values of the Average response available time, Average DNS lookup time, and Average TCP connection time measures, administrators can swiftly and accurately figure out the exact reason why there was a delay in receiving the first response byte. If this comparison reveals that the *Average DNS lookup time* is the highest, it implies that domain name resolution by the DNS server is taking a long time and impacting responsiveness. If the *Average TCP connection time* is found to be the culprit, then blame the network connection for delaying the transmission of the response byte. If the *Average reponse available time* is higher than the rest, you can be rest assured that the source of the problem lies with the server hosting the web site/web application.
Average response available time:	Indicates the interval between the start of	ms	In Figure 3.76, the *Average response available time* is the time spent between

Measurements	Description	Measurement Unit	Interpretation
	processing of a request on the browser on this device type to when this device type receives the response.		the *requestStart* event and *responseStart* event. Ideally, a low value is desired for this measure, as high values will certainly hurt the *Apdex score* of the web site/web application. The key factor that can influence the value of this measure is the request processing ability of the web server/web application server that is hosting the web site/web application being monitored. Any slowdown in the backend web server/web application server - caused by the lack of adequate processing power in or improper configuration of the backend server - can significantly delay request processing by the server. In its aftermath, the *Average response available time* will increase, leaving users with an unsatisfactory experience with the web site/web application.
Average server connection time:	Indicates the elapsed time since a user using this device type initiates a request and the start of fetching the response document from the server or application.	ms	In Figure 3.76, the time spent between navigationStart and requestStart makes up the Average server connection time. This includes the time to perform DNS lookups and the time to establish a TCP connection with the server. In other words, the value of this measure is nothing but the sum of the *Average DNS lookup time* and the *Average TCP connection time* measures. Ideally, the value of this measure should be low. A very high value will often end up delaying page loading and degrading the quality of the web site service. In the event that the server connection time is high therefore, simply compare the

Measurements	Description	Measurement Unit	Interpretation
			values of the *Average DNS lookup time* and *Average TCP connection time* measures to know to what this delay can be attributed - a delay in domain name resolution? Or a poor network connection to the server?
Average DNS lookup time:	Indicates the time taken by this device type to perform the domain lookup for connecting to the web site/web application.	ms	A high value for this measure will not only affect DNS lookup, but will also impact the *Average server connection time* and *Average page load time*. This naturally will have a disastrous effect on user experience.
Average TCP connection time:	Indicates the time taken by this device type to establish a TCP connection with the server.	ms	A bad network connection between the device and the server will result in delaying TCP connections to the server. As a result, the *Average server connection time* too will increase, thus impacting page load time and overall user experience with the web site/web application.

3.6.4 Page Types Test

Page types can be classified as follows:

- **Page**: This refers to the plain vanilla pages – i.e., the normal base pages.
- **iFrame**: iFr ames allow a visual HTML Browser window to be split into segments, each of which can show a different document.
- **AJAX**: AJAX (Asynchronous JavaScript and XML) allows web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes. This means that using AJAX, it is possible to update parts of a web page, without reloading the whole page.

A single web page in a web site/web application may contain more than one page type – in other words, it can contain a base page, one/more iFrames, and AJAX pages. Much against popular notion, base pages may not always be responsible for slowing down user accesses to a web site/web application. More often than not, iFrame URLs that that take hours to load and inefficient AJAX code, cause users to experience slowness or errors when accessing a web site/web application. This is why, when users complain that their web site/web application is responding slowly to requests, administrators need to rapidly determine whether the slowness is

owing to the base pages themselves, or because of iFrames and/or AJAX pages operating within the base pages. Accurate identification of the problem source enables administrators to figure out exactly what should be done to enhance performance of the web site/web application – should the base pages be re-engineered? should the iFrame URLs be pulled up for scrutiny? or should the AJAX code be cleaned up?

The **Page Types** test looks 'under-the-hood' of a web site/web application, discovers the page types that are in use, and reports the user experience per page type, so that administrators can instantly identify which page type is the least responsive or is error-prone. Detailed diagnostics of the test also lead you to the precise pages of a type that are slow, and what is causing the slowness – is it a slow front end? latent network? or a busy backend? The actual JavaScript errors that occurred in the pages of each type are also available as part of the detailed diagnosis, so as to facilitate easy and effective troubleshooting.

Target of the test : A web site/web application managed as a *Real User Monitor*

Agent deploying the test : A remote agent

Outputs of the test : One set of results for each page type

Test parameters:

TEST PERIOD	How often should the test be executed
PROXYHOST	If the eG agent communicates with the RUM collector via a proxy, then specify the IP address/fully-qualified host name of the proxy server here. By default, this is set to *none*, indicating that the eG agent does not communicate with the collector via a proxy.
PROXYPORT	If the eG agent communicates with the RUM collector via a proxy, then specify the port at which the proxy server listens for requests from the eG agent. By default, this is set to *none*, indicating that the eG agent does not communicate with the collector via a proxy.
PROXYUSERNAME and **PROXYPASSWORD**	If the eG agent communicates with the RUM collector via a proxy server, and if proxy server requires authentication, then specify valid credentials for authentication against **PROXYUSERNAME** and **PROXYPASSWORD**. If no proxy server is used or if the proxy server used does not require authentication, then set the **PROXYUSERNAME** and **PROXYPASSWORD** to *none*.
CONFIRM PASSWORD	Confirm the **PROXYPASSWORD** by retyping it here. **Note:** If you **Reconfigure** the test later to change the values of the **PROXYUSERNAME**, **PROXYPASSWORD**, and **CONFIRM PASSWORD** parameters, then such changes will be effected only if the eG remote agent monitoring the Real User Monitor component is restarted.
DO YOU WANT TO LIMIT THE PAGE VIEWS?	By default, the eG RUM monitors all requests to a managed web site. This is why, this flag is set to **No** by default. However, in case of web sites that receive thousands of hits every day, monitoring each page view may add significantly to the overhead of the eG agent and may also increase the size of the eG database considerably. To reduce the strain on both the eG agent and the eG backend, you may want to restrict the monitoring scope of this test to a few page visits. To achieve this, first set this flag to **Yes**. This will invoke the option depicted by the below figure.

	 Figure 3.80: Configuring the number of allowed page visits By default, the **MAXIMUM ALLOWED PAGE VISITS PER DAY** is set to *100000*. This implies that the test will consider only the first 100000 requests in a day for monitoring. All page visits beyond 100000 will by default be excluded from the test's monitoring purview. You can increase or decrease this limit, if you so need.
URL SEGMENTS TO BE USED AS GROUPED URL	**This parameter is applicable to the Page Groups test alone**. The **Page Groups** test groups URLs based on the URL segments configured for monitoring and reports aggregated response time metrics for every group. Using this parameter, you can specify a comma-separated list of URL segment numbers based on which the pages are to be grouped. URL segments are the parts of a URL (after the base URL) or path delimited by slashes. So if you had the URL: http://www.eazykart.com/web/shopping/login.jsp , then http://www.eazykart.com will be the base URL or domain, */web* will be the first URL segment, */shopping* will be the second URL segment, and */login.jsp* will be the third URL segment. By default, this parameter is set to *1,2.* This default setting, when applied to the sample URL provided above, implies that the eG agent will aggregate request and response time metrics to all instrumented web pages (i.e., web pages with the code snippet) under the URL */web/shopping*. Here, */web* corresponds to the specification 1 (URL segment 1) and */shopping* corresponds to the specification 2 (URL segment 2) in the default value *1,2*. This in turn means that, if the web site consists of pages such as http://www.eazykart.com/web/shopping/products.jsp , http://www.eazykart.com/web/shopping/products/travel/bags.jsp , http://www.eazykart.com/web/shopping/payment.jsp, etc., then the eG agent will track the requests to and responses from all these web pages, aggregate the results, and present the aggregated metrics for the descriptor /web/shopping. This way, the test will create different page groups based on each of the second-level URL segments in the managed web site – eg., */web/movies* , */web/travel* , */web/reservations* , */partner/contacts* etc. – and will report aggregated metrics for each group so created. If you want, you can override the default setting by providing different URL segment numbers here. For instance, your specification can be just *2*. In this case, for the URL http://www.eazykart.com/web/shopping/login.jsp, the test will report metrics for the descriptor */shopping.* You can even set this parameter to 1,3. For a web site that contains URLs such as http://www.eazykart.com/web/shopping/products.jsp , http://www.eazykart.com/web/shopping/products/travel/bags.jsp , and http://www.eazykart.com/web/shopping/payment.jsp, this specification will result in the following descriptors: */web/products*, */web/products.jsp*, and */web/payment.jsp*.
URL PATTERNS TO BE IGNORED FROM	By default, this test does not track requests to the following URL patterns: **.js,*.css,*.jpeg,*.jpg,*.png*. If required, you can remove one/more patterns from this default list, so that such patterns are monitored, or can append more patterns to this list

MONITORING	in order to exclude them from monitoring. For instance, to additionally ignore URLs that end with *.gif* and *.bmp* when monitoring, you need to alter the default specification as follows: **.js,*.css,*.jpeg,*.jpg,*.png,*.gif,*.bmp* **Note:** The URL patterns configured here are not just used by the eG agent to filter out unimportant performance data during metrics collection; these patterns are also used by the RUM collector to determine which beacons should be accepted and which ones need to be discarded. Typically, the very first time this test runs and polls the RUM collector for metrics, the eG agent executing this test searches the performance records stored in the collector for data that pertains to the URL patterns configured for exclusion. If such data is found, the agent then ignores that data during metrics collection. This means that during the very first test execution, URL filtering is performed only by the eG agent. During this time, the RUM collector downloads the URL patterns configured against this parameter from the eG agent. Armed with this information, the collector then scans all beacons that browsers send subsequently, and determines if there are any beacons for the excluded URL patterns. If such beacons are found, the collector discards them instantly. Filtering URLs at the collector-level significantly reduces the load on the collector and conserves storage space on the collector; it also minimizes the workload of the eG agent. By additionally filtering URLs at the agent-level, eG makes sure that even if beacons pertaining to excluded URL patterns find their way into the RUM collector, they are captured and siphoned out by the eG agent.
JAVASCRIPT ERRORS TO BE IGNORED	By default, this test alerts administrators to all Javascript errors that occur in the monitored web site/web application. This is why, this parameter is set to *none* by default. Sometimes however, administrators may not want to be notified when certain types of Javascript errors occur – this could be because such errors are harmless or are a normal occurrence in their environment. In such circumstances, you can instruct the eG agent to ignore these errors when monitoring. For this, specify the Java script error message to be ignored in the **JAVASCRIPT ERRORS TO BE IGNORED** text box, in the following format: *<Javascript eror message>:<URL of the page/file where the message originated>*. For instance, say that the *login.html* page in your web site runs a few Java scripts that throw *Object expected* errors, which you want the eG agent to ignore. In this case, your error specification can be as follows: *Object expected:http://www.eazykart.com/web/login.html*. Alternatively, you can provide only that text string with which the error message begins in your specification – eg., *Object:http://www.eazykart.com/web/login.html*. Moreover, instead of the complete URL, you can specify just the name of the HTML/jsp/aspx page in which the error is to be ignored – example: *Object:login.html*. Sometimes, the individual web pages in your web site may not run any Java script directly. Instead, these web pages may include links to Java script files that will run the Java script and return the output to the web pages. If you want the eG agent to ignore

	certain errors thrown by such a Javascript file, then your error pattern specification should include the URL of the Javascript file and not the web page that points to it. This is because, in this case, the file is where the error message originates. For instance, in the same example above, if the *login.html* page points to a *validate.js* file, and you want to ignore the *Object expected errors* that this JS file throws, your error pattern specification will either be, *Object expected:validate.js*, or *Object:validate.js*. Multiple error message-URL combinations can also be provided as a comma-separated list. The format of your specification will be: <Javascript error message 1>:<Originating URL 1>,<Javascript error message 2>:<Originating URL 2>,. . . For example, to ignore the *Object expected* and *Uncaught TypeError* errors in the *login.html* page, use the following specification: *Object:login.html,Uncaught:login.html* Likewise, to ignore the *Object expected* error in the *login.html* page and the *Uncaught TypeError* in the *validate.js* file, your specification will be: *Object:login.html,Uncaught:validate.js* If you want to ignore the *Uncaught TypeError* across all the pages of the web site, your specification will be as follows: *Uncaught TypeError:All* **Note:** When specifying the *<Javascript error message>* to be ignored, take care of the following: • The error message should not contain any special characters – in particular, the ':' (the colon) and the ',' (the comma) characters should be avoided. • The case of the actual error message and the one specified as part of your error specification should match. This is because, the eG agent performs *case-sensitive* pattern matching.
MAXIMUM HEALTHY TRANSACTIONS IN DD	By default, this parameter is set to *5* indicating that the detailed diagnosis of this test will collect and display metrics related to the top-5 normal/healthy page views, in terms of the *Average page load time*. To identify the top 5, the eG agent sorts all healthy transactions in the ascending order of their *Average page load time*, and picks the first 5 transactions from this sorted list for display in the **DETAILED DIAGNOSIS** page and for storing in the eG database. You can however, increase or decrease this value depending upon how many healthy transactions you want to see in your detailed diagnosis, and how well-tuned or well-sized the eG database is. If you do not want the detailed diagnosis to include any healthy transaction, set the value of this parameter to *0*. To view all healthy transactions, set the value of this

	parameter to *all*. Before setting the value of this parameter to *all*, make sure that you have a well-sized eG database in place, as this setting will store details of every transaction that registers an *Avg page load time* value that is lower than the **SLOW TRANSACTION CUTOFF** specification. On a good day, this can result in numerous transactions, and can consume considerable space in the eG database.
MAXIMUM SLOW TRANSACTIONS IN DD	By default, this parameter is set to *5* indicating that the detailed diagnosis of this test will display metrics related to all the top-5 slow transactions, in terms of the *Average page load time*. To identify the top 5, the eG agent sorts all slow transactions in the descending order of their *Average page load time*, and picks the first 5 transactions from this sorted list for display in the **DETAILED DIAGNOSIS** page and for storing in the eG database. You can however, increase or decrease this value depending upon how many slow transactions you want to see in your detailed diagnosis, and how well-tuned or well-sized the eG database is. If you do not want the detailed diagnosis to include any slow transaction, set the value of this parameter to *0*. To view all slow transactions, set the value of this parameter to *all*. Before setting the value of this parameter to *all*, make sure that you have a well-sized eG database in place, as this setting will store details of every transaction that registers an *Avg page load time* value that is higher than the **SLOW TRANSACTION CUTOFF** specification. On a bad day, this can result in numerous transactions, and can consume considerable space in the eG database.
MAXIMUM ERROR TRANSACTIONS IN DD	By default, this parameter is set to *5*, indicating that the detailed diagnosis of this test will display metrics related to the top-5 transactions that encountered JavaScript errors, based on when those errors occurred. To identify the top 5, the eG agent sorts all error transactions in the descending order of the date/time at which the errors occurred, and picks the first 5 transactions from this sorted list for display in the **DETAILED DIAGNOSIS** page and for storing in the eG database. You can however, increase or decrease this value depending upon how many error transactions you want to see in your detailed diagnosis, and how well-tuned or well-sized the eG database is. If you do not want the detailed diagnosis to include any error transaction, set the value of this parameter to *0*. To view all error transactions, set the value of this parameter to *all*. Before setting the value of this parameter to *all*, make sure that you have a well-sized eG database in place, as this setting will store details of every transaction that encounters a JavaScript error. On a bad day, this can result in numerous transactions, and can consume considerable space in the eG database.
SLOW TRANSACTION CUTOFF (MS)	This test reports the count of slow page views and also pinpoints the pages that are slow. To determine whether/not a page is slow, this test uses the **SLOW TRANSACTION CUTOFF** parameter. By default, this parameter is set to *4000 millisecs* (i.e., 4 seconds). This means that, if a page takes more than 4 seconds to load, this test will consider that page as a slow page by default. You can increase or decrease this slow transaction cutoff according to what is 'slow' and what is 'normal' in your environment.

	Note: The default value of this parameter is the same as the default *Maximum threshold* setting of the *Avg page load time* measure – i.e., both are set to *4000 millisecs* by default. While the former helps eG to distinguish between slow and healthy page views for the purpose of providing detailed diagnosis, the latter tells eG when to generate an alarm on *Avg page load time*. For best results, it is recommended that both these settings are configured with the same value at all times. Therefore, if you change the value of one of these configurations, then make sure you update the value of the other as well. For instance, if the **SLOW TRASACTION CUTOFF** is changed to *6000 millisecs*, change the *Maximum Threshold* of the *Avg page load time* measure to *6000* millsecs as well.
SEND ZERO VALUES WHEN THERE IS NO TRAFFIC	By default, this flag is set to **No**. This implies that, if there is no traffic to a monitored web site/web application – i.e., if all measures of this test return only the value *0* - then the eG agent will not report these metrics to the eG manager. This also means that, by default, users to the eG monitoring console will not know that there is no traffic to the web site/web application. You can however, ensure that users to the eG monitoring console are informed of the absence of any user activity on the web site/web application. For this, set this flag to **Yes**. If this is done, then the eG agent will report all the metrics of this test to the eG manager, despite the fact that their value is *0*. These zero values will clearly indicate to users that there no traffic to the monitored web site/web application.
PAGELOADTIME CUTOFF (MS)	eG RUM relies on the Navigation API to track page views and capture page load time. Sometimes, the Navigation API can incorrectly report abnormally high values for page load time. To ensure that the eG agent ignores beacons with such page load time values, you can configure in this text box, the maximum value for page load time. By default, this is set to 3600000 (ms). This implies that by default, the eG agent will disregard all beacons that carry page load time values higher than 3600000 milliseconds. You can change this value to suit your environment.
PAGE TYPES TO BE INCLUDED IN DASHBOARD	By default, the eG RUM Dashboard displays details of base page views only. You can optionally include Ajax and/or iFrame views as well in the dashboard, by selecting the relevant options from this list box.
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**.
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.

Measurements made by the test

Measurement	Measurement	Measurement Unit	Interpretation
Page views:	Indicates the total number of times pages of this type were viewed by users to the web site/web application.	Number	This is a good measure of the traffic to a specific page type. Sudden, but significant spikes in the page view count could be a cause for concern, as it could be owing to a malicious virus attack or an unscrupulous attempt to hack your web site/web application. **Note:** An abnormally high value for this measure may not always be a cause for concern; nor would it always indicate a genuine increase in traffic to the web site/web application. If the eG agent monitoring the *Real User Monitor* component is stopped for sometime (say, for maintenance purposes) and then started, or if the eG agent-collector connection breaks and is restored after a while, the eG agent will pull all the metrics that the collector stored locally during the period of its absence, cumulate them, and then display the cumulated values in the eG monitoring console as metrics that pertain to the current measurement period. In reality, these metrics pertain to the entire time period when the eG agent was unavailable. Because of this, the *Page views* measure may indicate a sudden and significant surge in traffic.

Measurement	Measurement	Measurement Unit	Interpretation
Apdex score:	Indicates the apdex score of the web site/web application based on the experience of users to this page type.	Number	**Apdex (Application Performance Index)** is an open standard developed by an alliance of companies. It defines a standard method for reporting and comparing the performance of software applications in computing. Its purpose is to convert measurements into insights about user satisfaction, by specifying a uniform way to analyze and report on the degree to which measured performance meets user expectations. The Apdex method converts many measurements into one number on a uniform scale of 0-to-1 (0 = no users satisfied, 1 = all users satisfied). The resulting Apdex score is a numerical measure of user satisfaction with the performance of enterprise applications. This metric can be used to report on any source of end- user performance measurements for which a performance objective has been defined. The Apdex formula is: $Apdex_t$ = *(Satisfied Count + Tolerating Count / 2) / Total Samples* This is nothing but the number of satisfied samples plus half of the tolerating samples plus none of the frustrated samples, divided by all the samples. A score of 1.0 means all responses were satisfactory. A score of 0.0 means none of the responses were satisfactory. Tolerating responses half satisfy a user. For example, if all responses are tolerating, then the Apdex score would

Measurement	Measurement	Measurement Unit	Interpretation
			be 0.50. Ideally therefore, the value of this measure should be *1.0*. A value less than *1.0* indicates that the experience of users to this page type has been less than satisfactory.
Average page load time:	Indicates the average time taken by the pages of this type to load completely on the browser.	ms	This is the average interval between the time that a user initiates a request and the completion of the page load of the response in the user's browser. In the context of an Ajax request, it ends when the response has been completely processed. By comparing the value of this measure across page types, you will be able to tell if the page load time is significantly higher for any one type of page – this could be the page type that is causing the slowness. You may want to compare the values of the of the *Average front end time*, *Average server connection time*, and *Average response available time* measures for that page type, to know what is exactly causing pages of that type to load slowly – is it the front end? network? or the backend? To know which pages of the type are slow, use the detailed diagnosis of this measure.
Unique user session:	**Unique user session:** Indicates the number of distinct users who are currently accessing pages of this type in the web site/web application.	Number	

Measurement	Measurement	Measurement Unit	Interpretation
Page views per minute:	Indicates the number of times the pages of this type were viewed per minute.	Number	An unusually high value for this measure may require investigation.
Normal page view percentage:	Indicates the percentage of page views of this type that delivered a satisfactory experience to users.	Percent	The value of this measure indicates the percentage of page views of this type in which users have neither experienced any slowness, nor encountered any Javascript errors. Ideally, the value of this measure should be 100%. A value that is slightly less than 100% indicates that the user experience with pages of this type has not been up to the mark. A value less than 50% is indicative of a serious problem, where most of the page views of this type are either slow or have encountered Javascript errors. Under such circumstances, to know what exactly is affecting the experience of users to this page type, compare the value of the *Slow page view percentage* with that of the *Javasript error page view percentage* for that browser. This will reveal the reason for the poor user experience – slow pages? or Javascript errors? If slow pages are the problem, use the detailed diagnosis of the *Slow page views* measure to know which pages of that type are slow and where these pages are losing time – in the front end? network? or backend?. If JavaScript errors are the problem, use the detailed diagnosis of the *JavaScript error view percentage* measure to know what errors occurred in which pages of the type.

Measurement	Measurement	Measurement Unit	Interpretation
Slow page view percentage:	Indicates the percentage of page views of this type that are slow in loading.	Percent	Ideally, the value of this measure should be 0. A value over 50% implies that you are in a spot of bother, with over half of the page views being slow. Use the detailed diagnosis of the *Slow page views* measure to identify the slow pages and isolate the root- cause of the slowness – is it the front end? the network? or the backend?
JavaScript error view percentage:	Indicates the percentage of page views of this type that have encountered JavaScript errors.	Percent	Ideally, the value of this measure should be 0. A value over 50% implies that you are in a spot of bother, with over half of the page views of this type are experiencing JavaScript errors.
Slow page views (Tolerating & Frustrated):	Indicates the number of page views of this type that were slow.	Number	A page view is considered to be slow when the average time taken to load that page exceeds the **SLOW TRANSACTION CUTOFF** configured for this test. Ideally, a page should load quickly. The value 0 is hence desired for this measure. If the value of this measure is high, it indicates that users frequently experienced slowness when accessing pages of this type. To know which page views of this type are slow and why, use the detailed diagnosis of this measure.
JavaScript error page views:	Indicates the number of times JavaScript errors occurred when viewing pages of this type.	Number	Ideally, the value of this measure should be 0. A high value indicates that many JavaScript errors are occurring when viewing pages of this type. Use the detailed diagnosis of this measure to identify the error pages and to know what Javascript error has occurred in which page. This will greatly aid troubleshooting!

Measurement	Measurement	Measurement Unit	Interpretation
Satisfied page views:	Indicates the number of times pages of this type were viewed without any slowness.	Number	A page view is considered to be slow when the average time taken to load that page exceeds the **SLOW TRANSACTION CUTOFF** configured for this test. If this **SLOW TRANSACTION CUTOFF** is not exceeded, then the page view is deemed to be 'satisfactory'. To know which page views are satisfactory, use the detailed diagnosis of this measure. Ideally, the value of this measure should be the same as that of the *Page views* measure. If not, then it indicates that one/more page views are slow – i.e., have violated the **SLOW TRANSACTION CUTOFF**. If the value of this measure is much lesser than the value of the *Tolerating page views* and the *Frustrated page views*, it is a clear indicator that the user experience with this page type has been b elow- par. In such a case, use the detailed diagnosis of the *Tolerating page views* and *Frustrated page views* measures to know which pages are slow and why.
Tolerating page views:	Indicates the number of tolerating page views of this type.	Number	If the *Average page load time* of a page exceeds the **SLOW TRANSACTION CUTTOFF** configuration of this test, but is less than 4 times the **SLOW TRANSACTION CUTOFF** (i.e., < 4 * **SLOW TRANSACTION CUTOFF**), then such a page view is considered to be a **Tolerating page view**. Ideally, the value of this measure should be 0. A value higher than that of the *Satisfied page views* measure is a cause for concern, as it implies that the overall

Measurement	Measurement	Measurement Unit	Interpretation
			user experience with this page type is less than satisfactory. To know which pages are contributing to this sub-par experience, use the detailed diagnosis of this measure. The detailed metrics will also enable you to accurately isolate what is causing the tolerating page views – a problem with the front end? network? or backend?
Frustrated page views:	Indicates the number of frustrated page views of this type.	Number	If the *Average page load time* of a page is over 4 times the **SLOW TRANSACTION CUTTOFF** configuration of this test (i.e., > 4 * **SLOW TRANSACTION CUTOFF**), then such a page view is considered to be a **Frustrated page view**. Ideally, the value of this measure should be 0. A value higher than that of the *Satisfied page views* measure is a cause for concern, as it implies that the experience of users to this page type has been less than satisfactory. To know which pages are contributing to this sub-par experience, use the detailed diagnosis of this measure. The detailed metrics will also enable you to accurately isolate what is causing the frustrated page views – a problem with the front end? network? or backend?
Average front end time:	Indicates the interval between the arrival of the first byte of text response and the completion of the response page rendering by the browser for this page type.	ms	In the Figure 3.76 that depicts a typical page loading process, the *Average front end time* denotes the time from the *responseStart* event to the *loadEventEnd* . This process includes document downloading, processing, and page rendering. This time is therefore the sum of the *Average DOM ready time* and the *Average page rendering time*.

Measurement	Measurement	Measurement Unit	Interpretation
			If the *Average page load time* exceeds its threshold, then you may want to compare the value of this measure with that of the *Average server connection time* and *Average response available time* to zoom into the source of the slowness – is it the front end? the network? or the backend?
Average page rendering time:	Indicates the time taken to complete the download of remaining resources, including images, and to finish rendering the pages of type in the browser.	ms	A high value of this measure indicates that the pages of this type are taking too long to be rendered. This can adversely impact the *Average front end time*, which in turn can prolong the *Average page load time*. Ideally therefore, the value of this measure should be low.
Average DOM ready time:	Indicates the time taken by the browser to make the complete HTML document (DOM) available for JavaScript to apply rendering logic on the pages of this type.	ms	The value of this measure is the sum of the *Average DOM download time* and the *Average DOM processing time* measures. If the value of this measure is very high, then you may want to compare the *Average DOM download time* and the *Average DOM processing time* measures to figure out what is delaying DOM building – downloading? Or processing? A high value for this measure can adversely impact the *Average front end time* , which in turn can prolong the *Average page load time*. Ideally therefore, the value of this measure should be low.
Average DOM download time:	Indicates the time taken to download the complete HTML document for this page type on the browser.	ms	Higher the download time of the document, longer will be the time taken to make the document available for page rendering. As a result, the overall user experience will be affected! This is why, a low value is desired for this measure at all times.

Measurement	Measurement	Measurement Unit	Interpretation
Average DOM processing time:	Indicates the time taken by the browser to build the Document Object Model (DOM) for the pages of this type and make it available for JavaScript to apply rendering logic.	ms	An unusually high value for this measure is a clear indicator that DOM building is taking longer than normal. In consequence, page rendering will be delayed, thus adversely impacting user experience with pages of this type. Ideally therefore, the value of this measure should be low.
Average first byte time:	Indicates the interval between the time that a user initiates a request and the time that the browser receives the first response byte for this page type. In the context of an Ajax request, this is the interval between the Ajax request dispatch and the time that the browser receives the first response byte.	ms	Going by Figure 3.76 above, the *Average first byte time* is the time that elapsed between *navigationStart* and *responseStart*. The value of this measure is also the sum of *Average response available time*, *Average DNS lookup time*, and *Average TCP connection time*. This means that an abnormal increase in any of the above-mentioned time values will increase the value of this measure. If the first response byte from the target web site/web application is itself received slowly, it is bound to have a cascading effect on all events that follow – such as, document downloading, processing, and page rendering. Ultimately, this will impact the page load time as seen by end-users. This is why, if the *Average first byte time* violates its threshold, administrators need to instantly switch to the troubleshooting mode and rapidly isolate what is causing it – is DNS lookup taking a long time? is the network connection to the web site/web application latent? or is the web server/web application server hosting the web site slow in processing requests? By comparing the values of the *Average response available time*, *Average DNS lookup time*, and *Average TCP connection time* measures,

Measurement	Measurement	Measurement Unit	Interpretation
			administrators can swiftly and accurately figure out the exact reason why there was a delay in receiving the first response byte. If this comparison reveals that the *Average DNS lookup time* is the highest, it implies that domain name resolution by the DNS server is taking a long time and impacting responsiveness. If the *Average TCP connection time* is found to be the culprit, then blame the network connection for delaying the transmission of the response byte. If the *Average reponse available time* is higher than the rest, you can be rest assured that the source of the problem lies with the server hosting the web site/web application.
Average response available time:	Indicates the interval between the start of processing of a request on the browser for this page type to when the response is received.	ms	In Figure 3.76, the *Average response available* time is the time spent between the *requestStart* event and *responseStart* event. Ideally, a low value is desired for this measure, as high values will certainly hurt the *Apdex scrore* of the web site/web application. The key factor that can influence the value of this measure is the request processing ability of the web server/web application server that is hosting the web site/web application being monitored. Any slowdown in the backend web server/web application server – caused by the lack of adequate processing power in or improper configuration of the backend server – can significantly delay request processing by the server. In its aftermath, the *Average response*

Measurement	Measurement	Measurement Unit	Interpretation
			available time will increase, leaving users with an unsatisfactory experience with the web site/web application.
Average server connection time:	Indicates the elapsed time since a user initiates a request to this page type and the start of fetching the response document from it.	ms	In Figure 3.76, the time spent between *navigationStart* and *requestStart* makes up the *Average server connection time*. This includes the time to perform DNS lookups and the time to establish a TCP connection with the server. In other words, the value of this measure is nothing but the sum of the *Average DNS lookup time* and the *Average TCP connection time* measures. Ideally, the value of this measure should be low. A very high value will often end up delaying page loading and degrading the quality of the web site service. In the event that the server connection time is high therefore, simply compare the values of the *Average DNS lookup time* and *Average TCP connection time* measures to know to what this delay can be attributed – a delay in domain name resolution? Or a poor network connection to the server?
Average DNS lookup time:	Indicates the time taken by this page type to perform the domain lookup for connecting to the web site/web application.	ms	A high value for this measure will not only affect DNS lookup, but will also impact the *Average server connection time* and *Average page load time*. This naturally will have a disastrous effect on user experience.
Average TCP connection time:	Indicates the time taken by this page type to establish a TCP connection with the server.	ms	A bad network connection between the user's browser and the server will result in delaying TCP connections to the server. As a result, the *Average server connection time* too will increase, thus

Measurement	Measurement	Measurement Unit	Interpretation
			impacting page load time and overall user experience with the web site/web application.

3.6.5 Page Groups Test

The **Page Groups** test groups URLs based on the URL segments configured for monitoring and reports aggregated respon se time metrics for every group. URL segments are the parts of a URL (after the base URL) or path delimited by slashes. So if you had the URL: http://www.eazykart.com/web/shopping/login.jsp, then http://www.eazykart.com will be the base URL or domain, */web* will be the first URL segment, */shopping* will be the second URL segment, and */login.jsp* will be the third URL segment.

If you know of specific URL segments in your web site/web application that typically handle critical business transactions – eg., login, shopping, payment etc. – you may want to configure such segments for monitoring using the **Page Groups** test. The test monitors the time taken by the pages in each group to load, reports the average load time of the group as a whole, and thus leads you to the broad page groups that are adversely impacting user experience with the web site/web application. This way, you can identify what type of transactions to the web site/web application are slow. Detailed diagnostics provided by the test also lead you to the precise pages in the group that are slow and the reason for the slowness – is it the front end? network? or back end?

Target of the test : A web site/web application managed as a *Real User Monitor*

Agent deploying the test : A remote agent

Outputs of the test : One set of results for each browser used for accessing the web site/web application being monitored

Test parameters:

Configurable parameters for the test

TEST PERIOD	How often should the test be executed
PROXYHOST	If the eG agent communicates with the RUM collector via a proxy, then specify the IP address/fully-qualified host name of the proxy server here. By default, this is set to *none*, indicating that the eG agent does not communicate with the collector via a proxy.
PROXYPORT	If the eG agent communicates with the RUM collector via a proxy, then specify the port at which the proxy server listens for requests from the eG agent. By default, this is set to *none*, indicating that the eG agent does not communicate with the collector via a proxy.
PROXYUSERNAME and	If the eG agent communicates with the RUM collector via a proxy server, and if proxy server requires authentication, then specify valid credentials for authentication against

PROXYPASSWORD	**PROXYUSERNAME** and **PROXYPASSWORD**. If no proxy server is used or if the proxy server used does not require authentication, then set the **PROXYUSERNAME** and **PROXYPASSWORD** to *none*.
CONFIRM PASSWORD	Confirm the **PROXYPASSWORD** by retyping it here. **Note:** If you **Reconfigure** the test later to change the values of the **PROXYUSERNAME**, **PROXYPASSWORD**, and **CONFIRM PASSWORD** parameters, then such changes will be effected only if the eG remote agent monitoring the Real User Monitor component is restarted.
DO YOU WANT TO LIMIT THE PAGE VIEWS?	By default, the eG RUM monitors all requests to a managed web site. This is why, this flag is set to **No** by default. However, in case of web sites that receive thousands of hits every day, monitoring each page view may add significantly to the overhead of the eG agent and may also increase the size of the eG database considerably. To reduce the strain on both the eG agent and the eG backend, you may want to restrict the monitoring scope of this test to a few page visits. To achieve this, first set this flag to **Yes**. This will invoke the option depicted by the below figure.  Figure 3.81: Configuring the number of allowed page visits By default, the **MAXIMUM ALLOWED PAGE VISITS PER DAY** is set to *100000*. This implies that the test will consider only the first 100000 requests in a day for monitoring. All page visits beyond 100000 will by default be excluded from the test's monitoring purview. You can increase or decrease this limit, if you so need.
URL SEGMENTS TO BE USED AS GROUPED URL	**This parameter is applicable to the Page Groups test alone**. The **Page Groups** test groups URLs based on the URL segments configured for monitoring and reports aggregated response time metrics for every group. Using this parameter, you can specify a comma-separated list of URL segment numbers based on which the pages are to be grouped. URL segments are the parts of a URL (after the base URL) or path delimited by slashes. So if you had the URL: http://www.eazykart.com/web/shopping/login.jsp , then http://www.eazykart.com will be the base URL or domain, */web* will be the first URL segment, */shopping* will be the second URL segment, and */login.jsp* will be the third URL segment. By default, this parameter is set to *1,2*. This default setting, when applied to the sample URL provided above, implies that the eG agent will aggregate request and response time metrics to all instrumented web pages (i.e., web pages with the code snippet) under the URL */web/shopping*. Here, */web* corresponds to the specification 1 (URL segment 1) and */shopping* corresponds to the specification 2 (URL segment 2) in the default value *1,2*. This in turn means that, if the web site consists of pages such as http://www.eazykart.com/web/shopping/products.jsp , http://www.eazykart.com/web/shopping/products/travel/bags.jsp ,

	http://www.eazykart.com/web/shopping/payment.jsp, etc., then the eG agent will track the requests to and responses from all these web pages, aggregate the results, and present the aggregated metrics for the descriptor */web/shopping*. This way, the test will create different page groups based on each of the second-level URL segments in the managed web site – eg., */web/movies* , */web/travel* , */web/reservations* , */partner/contacts* etc. – and will report aggregated metrics for each group so created. If you want, you can override the default setting by providing different URL segment numbers here. For instance, your specification can be just *2*. In this case, for the URL http://www.eazykart.com/web/shopping/login.jsp, the test will report metrics for the descriptor */shopping.* You can even set this parameter to 1,3. For a web site that contains URLs such as http://www.eazykart.com/web/shopping/products.jsp , http://www.eazykart.com/web/shopping/products/travel/bags.jsp , and http://www.eazykart.com/web/shopping/payment.jsp, this specification will result in the following descriptors: */web/products*, */web/products.jsp*, and */web/payment.jsp*.
URL PATTERNS TO BE IGNORED FROM MONITORING	By default, this test does not track requests to the following URL patterns: **.js,*.css,*.jpeg,*.jpg,*.png*. If required, you can remove one/more patterns from this default list, so that such patterns are monitored, or can append more patterns to this list in order to exclude them from monitoring. For instance, to additionally ignore URLs that end with *.gif* and *.bmp* when monitoring, you need to alter the default specification as follows: **.js,*.css,*.jpeg,*.jpg,*.png,*.gif,*.bmp* **Note**: The URL patterns configured here are not just used by the eG agent to filter out unimportant performance data during metrics collection; these patterns are also used by the RUM collector to determine which beacons should be accepted and which ones need to be discarded. Typically, the very first time this test runs and polls the RUM collector for metrics, the eG agent executing this test searches the performance records stored in the collector for data that pertains to the URL patterns configured for exclusion. If such data is found, the agent then ignores that data during metrics collection. This means that during the very first test execution, URL filtering is performed only by the eG agent. During this time, the RUM collector downloads the URL patterns configured against this parameter from the eG agent. Armed with this information, the collector then scans all beacons that browsers send subsequently, and determines if there are any beacons for the excluded URL patterns. If such beacons are found, the collector discards them instantly. Filtering URLs at the collector-level significantly reduces the load on the collector and conserves storage space on the collector; it also minimizes the workload of the eG agent. By additionally filtering URLs at the agent-level, eG makes sure that even if beacons pertaining to excluded URL patterns find their way into the RUM collector, they are captured and siphoned out by the eG agent.
JAVASCRIPT ERRORS TO BE IGNORED	By default, this test alerts administrators to all Javascript errors that occur in the monitored web site/web application. This is why, this parameter is set to *none* by default. Sometimes however, administrators may not want to be notified when certain

	types of Javascript errors occur – this could be because such errors are harmless or are a normal occurrence in their environment. In such circumstances, you can instruct the eG agent to ignore these errors when monitoring. For this, specify the Java script error message to be ignored in the JAVASCRIPT **ERRORS TO BE IGNORED** text box, in the following format: *<Javascript eror message>:<URL of the page/file where the message originated>*. For instance, say that the *login.html* page in your web site runs a few Java scripts that throw *Object expected* errors, which you want the eG agent to ignore. In this case, your error specification can be as follows: *Object expected:http://www.eazykart.com/web/login.html*. Alternatively, you can provide only that text string with which the error message begins in your specification – eg., *Object:http://www.eazykart.com/web/login.html*. Moreover, instead of the complete URL, you can specify just the name of the HTML/jsp/aspx page in which the error is to be ignored – example: *Object:login.html*. Sometimes, the individual web pages in your web site may not run any Java script directly. Instead, these web pages may include links to Java script files that will run the Java script and return the output to the web pages. If you want the eG agent to ignore certain errors thrown by such a Javascript file, then your error pattern specification should include the URL of the Javascript file and not the web page that points to it. This is because, in this case, the file is where the error message originates. For instance, in the same example above, if the *login.html* page points to a *validate.js* file, and you want to ignore the *Object expected errors* that this JS file throws, your error pattern specification will either be, *Object expected:validate.js*, or *Object:validate.js*. Multiple error message-URL combinations can also be provided as a comma-separated list. The format of your specification will be: <Javascript error message 1>:<Originating URL 1>,<Javascript error message 2>:<Originating URL 2>,. . . For example, to ignore the *Object expected* and *Uncaught TypeError* errors in the *login.html* page, use the following specification: Object:login.html,Uncaught:login.html Likewise, to ignore the *Object expected* error in the *login.html* page and the *Uncaught TypeError* in the *validate.js* file, your specification will be: *Object:login.html,Uncaught:validate.js* If you want to ignore the *Uncaught TypeError* across all the pages of the web site, your specification will be as follows: Uncaught TypeError:All **Note**: When specifying the *<Javascript error message>* to be ignored, take care of the following:

	• The error message should not contain any special characters – in particular, the ':' (the colon) and the ',' (the comma) characters should be avoided. • The case of the actual error message and the one specified as part of your error specification should match. This is because, the eG agent performs *case-sensitive* pattern matching.
MAXIMUM HEALTHY TRANSACTIONS IN DD	By default, this parameter is set to 5 indicating that the detailed diagnosis of this test will collect and display metrics related to the top-5 normal/healthy page views, in terms of the *Average page load time*. To identify the top 5, the eG agent sorts all healthy transactions in the ascending order of their *Average page load time*, and picks the first 5 transactions from this sorted list for display in the **DETAILED DIAGNOSIS** page and for storing in the eG database. You can however, increase or decrease this value depending upon how many healthy transactions you want to see in your detailed diagnosis, and how well-tuned or well-sized the eG database is. If you do not want the detailed diagnosis to include any healthy transaction, set the value of this parameter to *0*. To view all healthy transactions, set the value of this parameter to *all*. Before setting the value of this parameter to *all*, make sure that you have a well-sized eG database in place, as this setting will store details of every transaction that registers an *Avg page load time* value that is lower than the **SLOW TRANSACTION CUTOFF** specification. On a good day, this can result in numerous transactions, and can consume considerable space in the eG database.
MAXIMUM SLOW TRANSACTIONS IN DD	By default, this parameter is set to *5* indicating that the detailed diagnosis of this test will display metrics related to all the top-5 slow transactions, in terms of the *Average page load time*. To identify the top 5, the eG agent sorts all slow transactions in the descending order of their *Average page load time*, and picks the first 5 transactions from this sorted list for display in the **DETAILED DIAGNOSIS** page and for storing in the eG database. You can however, increase or decrease this value depending upon how many slow transactions you want to see in your detailed diagnosis, and how well-tuned or well-sized the eG database is. If you do not want the detailed diagnosis to include any slow transaction, set the value of this parameter to *0*. To view all slow transactions, set the value of this parameter to *all*. Before setting the value of this parameter to *all*, make sure that you have a well-sized eG database in place, as this setting will store details of every transaction that registers an *Avg page load time* value that is higher than the **SLOW TRANSACTION CUTOFF** specification. On a bad day, this can result in numerous transactions, and can consume considerable space in the eG database.
MAXIMUM ERROR TRANSACTIONS IN DD	By default, this parameter is set to *5*, indicating that the detailed diagnosis of this test will display metrics related to the top-5 transactions that encountered JavaScript errors, based on when those errors occurred. To identify the top 5, the eG agent sorts all error transactions in the descending order of the date/time at which the errors occurred, and

	picks the first 5 transactions from this sorted list for display in the **DETAILED DIAGNOSIS** page and for storing in the eG database. You can however, increase or decrease this value depending upon how many error transactions you want to see in your detailed diagnosis, and how well-tuned or well-sized the eG database is. If you do not want the detailed diagnosis to include any error transaction, set the value of this parameter to *0*. To view all error transactions, set the value of this parameter to *all*. Before setting the value of this parameter to *all*, make sure that you have a well-sized eG database in place, as this setting will store details of every transaction that encounters a JavaScript error. On a bad day, this can result in numerous transactions, and can consume considerable space in the eG database.
SLOW TRANSACTION CUTOFF (MS)	This test reports the count of slow page views and also pinpoints the pages that are slow. To determine whether/not a page is slow, this test uses the **SLOW TRANSACTION CUTOFF** parameter. By default, this parameter is set to *4000 millisecs* (i.e., 4 seconds). This means that, if a page takes more than 4 seconds to load, this test will consider that page as a slow page by default. You can increase or decrease this slow transaction cutoff according to what is 'slow' and what is 'normal' in your environment. **Note**: The default value of this parameter is the same as the default *Maximum threshold* setting of the *Avg page load time* measure – i.e., both are set to *4000 millisecs* by default. While the former helps eG to distinguish between slow and healthy page views for the purpose of providing detailed diagnosis, the latter tells eG when to generate an alarm on *Avg page load time*. For best results, it is recommended that both these settings are configured with the same value at all times. Therefore, if you change the value of one of these configurations, then make sure you update the value of the other as well. For instance, if the **SLOW TRASACTION CUTOFF** is changed to *6000 millisecs*, change the *Maximum Threshold* of the *Avg page load time* measure to *6000* millsecs as well.
SEND ZERO VALUES WHEN THERE IS NO TRAFFIC	By default, this flag is set to **No**. This implies that, if there is no traffic to a monitored web site/web application – i.e., if all measures of this test return only the value *0* - then the eG agent will not report these metrics to the eG manager. This also means that, by default, users to the eG monitoring console will not know that there is no traffic to the web site/web application. You can however, ensure that users to the eG monitoring console are informed of the absence of any user activity on the web site/web application. For this, set this flag to **Yes**. If this is done, then the eG agent will report all the metrics of this test to the eG manager, despite the fact that their value is *0*. These zero values will clearly indicate to users that there no traffic to the monitored web site/web application.
PAGELOADTIME CUTOFF (MS)	eG RUM relies on the Navigation API to track page views and capture page load time. Sometimes, the Navigation API can incorrectly report abnormally high values for page load time. To ensure that the eG agent ignores beacons with such page load time

	values, you can configure in this text box, the maximum value for page load time. By default, this is set to 3600000 (ms). This implies that by default, the eG agent will disregard all beacons that carry page load time values higher than 3600000 milliseconds. You can change this value to suit your environment.
PAGE TYPES TO BE INCLUDED IN DASHBOARD	By default, the eG RUM Dashboard displays details of base page views only. You can optionally include Ajax and/or iFrame views as well in the dashboard, by selecting the relevant options from this list box.
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**.
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.

Measurements made by the test

Measurement	Measurement	Measurement Unit	Interpretation
Page views:	Indicates the total number of times pages in this group were viewed by users to the web site/web application.	Number	This is a good measure of the traffic to a specific page group. Sudden, but significant spikes in the page view count could be a cause for concern, as it could be owing to a malicious virus attack or an unscrupulous attempt to hack your web site/web application. **Note:** An abnormally high value for this measure may not always be a cause for concern; nor would it always indicate a

Measurement	Measurement	Measurement Unit	Interpretation
			genuine increase in traffic to the web site/web application. If the eG agent monitoring the *Real User Monitor* component is stopped for sometime (say, for maintenance purposes) and then started, or if the eG agent-collector connection breaks and is restored after a while, the eG agent will pull all the metrics that the collector stored locally during the period of its absence, cumulate them, and then display the cumulated values in the eG monitoring console as metrics that pertain to the current measurement period. In reality, these metrics pertain to the entire time period when the eG agent was unavailable. Because of this, the *Page views* measure may indicate a sudden and significant surge in traffic.
Apdex score:	Indicates the apdex score of the web site/web application based on the experience of users to this page group.	Number	**Apdex (Application Performance Index)** is an open standard developed by an alliance of companies. It defines a standard method for reporting and comparing the performance of software applications in computing. Its purpose is to convert measurements into insights about user satisfaction, by specifying a uniform way to analyze and report on the degree to which measured performance meets user expectations. The Apdex method converts many measurements into one number on a uniform scale of 0-to-1 (0 = no users satisfied, 1 = all users satisfied). The resulting Apdex score is a numerical measure of user satisfaction with the performance of enterprise applications. This metric can be used to report on any

Measurement	Measurement	Measurement Unit	Interpretation
			source of end- user performance measurements for which a performance objective has been defined. The Apdex formula is: *Apdex $_t$ = (Satisfied Count + Tolerating Count / 2) / Total Samples* This is nothing but the number of satisfied samples plus half of the tolerating samples plus none of the frustrated samples, divided by all the samples. A score of 1.0 means all responses were satisfactory. A score of 0.0 means none of the responses were satisfactory. Tolerating responses half satisfy a user. For example, if all responses are tolerating, then the Apdex score would be 0.50. Ideally therefore, the value of this measure should be *1.0*. A value less than *1.0* indicates that the experience of users to this page group has been less than satisfactory.
Average page load time:	Indicates the average time taken by the pages in this group to load completely on the browser.	ms	This is the average interval between the time that a user initiates a request and the completion of the page load of the response in the user's browser. In the context of an Ajax request, it ends when the response has been completely processed. By comparing the value of this measure across page groups, you will be able to tell if the page load time is significantly higher for any one group – this could be the page group that is causing the slowness. To know which pages in the group are slow, use the detailed

Measurement	Measurement	Measurement Unit	Interpretation
			diagnosis of this measure. Compare the values of the *Average front end time*, *Average server connection time*, and *Average response available time* measures for that page group, to know what is exactly causing pages of that type to load slowly – is it the front end? network? or the backend?
Unique user session:	Indicates the number of distinct users who are currently accessing pages in this group.	Number	
Page views per minute:	Indicates the number of times the pages in this group were viewed per minute.	Number	An unusually high value for this measure may require investigation.
Normal page view percentage:	Indicates the percentage of page views in this group that delivered a satisfactory experience to users.	Percent	The value of this measure indicates the percentage of page views in this group in which users have neither experienced any slowness, nor encountered any Javascript errors. Ideally, the value of this measure should be 100%. A value that is slightly less than 100% indicates that the user experience with pages of this type has not been up to the mark. A value less than 50% is indicative of a serious problem, where most of the page views in this group are either slow or have encountered Javascript errors. Under such circumstances, to know what exactly is affecting the experience of users to this page group, compare the value of the *Slow page view percentage* with that of the *Javasript error page view percentage* for that browser. This will reveal the

Measurement	Measurement	Measurement Unit	Interpretation
			reason for the poor user experience – slow pages? or Javascript errors? If slow pages are the problem, use the detailed diagnosis of the *Slow page views* measure to know which pages in the group are slow and where these pages are losing time – in the front end? network? or backend?. If JavaScript errors are the problem, use the detailed diagnosis of the *JavaScript error view percentage* measure to know what errors occurred in which pages of the type.
Slow page view percentage:	Indicates the percentage of page views in this group that are slow in loading.	Percent	Ideally, the value of this measure should be 0. A value over 50% implies that you are in a spot of bother, with over half of the page views in this group being slow. Use the detailed diagnosis of the *Slow page views* measure to identify the slow pages and isolate the root-cause of the slowness – is it the front end? the network? or the backend?
JavaScript error view percentage:	Indicates the percentage of page views in this group that have encountered JavaScript errors.	Percent	Ideally, the value of this measure should be 0. A value over 50% implies that you are in a spot of bother, with over half of the page views in this group experiencing JavaScript errors.
Slow page views (Tolerating & Frustrated):	Indicates the number of times pages in this page group took very long to be viewed .	Number	A page view is considered to be slow when the average time taken to load that page exceeds the **SLOW TRANSACTION CUTOFF** configured for this test. Ideally, a page should load quickly. The value 0 is hence desired for this measure. If the value of this measure is high, it indicates that users frequently

Measurement	Measurement	Measurement Unit	Interpretation
			experienced slowness when accessing pages in this group. To know which page views are slow and why, use the detailed diagnosis of this measure.
JavaScript error page views:	Indicates the number of times JavaScript errors occurred when viewing the pages in this group.	Number	Ideally, the value of this measure should be 0. A high value indicates that many JavaScript errors are occurring when viewing pages in this group. Use the detailed diagnosis of this measure to identify the error pages and to know what Javascript error has occurred in which page. This will greatly aid troubleshooting!
Satisfied page views:	Indicates the number of times pages in this group were viewed without any slowness.	Number	A page view is considered to be slow when the average time taken to load that page exceeds the **SLOW TRANSACTION CUTOFF** configured for this test. If this **SLOW TRANSACTION CUTOFF** is not exceeded, then the page view is deemed to be 'satisfactory'. To know which page views are satisfactory, use the detailed diagnosis of this measure. Ideally, the value of this measure should be the same as that of the *Page views* measure. If not, then it indicates that one/more page views are slow – i.e., have violated the **SLOW TRANSACTION CUTOFF**. If the value of this measure is much lesser than the value of the *Tolerating page views* and the *Frustrated page views*, it is a clear indicator that the user experience with this page group has been below-par. In such a case, use the detailed diagnosis of the *Tolerating page views* and *Frustrated page views* measures to know which pages are slow

Measurement	Measurement	Measurement Unit	Interpretation
			and why.
Tolerating page views:	Indicates the number of tolerating page views in this group.	Number	If the *Average page load time* of a page exceeds the **SLOW TRANSACTION CUTTOFF** configuration of this test, but is less than 4 times the **SLOW TRANSACTION CUTOFF** (i.e., < 4 * **SLOW TRANSACTION CUTOFF**), then such a page view is considered to be a **Tolerating page view**. Ideally, the value of this measure should be 0. A value higher than that of the *Satisfied page views* measure is a cause for concern, as it implies that the overall user experience with this page group is less than satisfactory. To know which pages are contributing to this sub-par experience, use the detailed diagnosis of this measure. The detailed metrics will also enable you to accurately isolate what is causing the tolerating page views – a problem with the front end? network? or backend?
Frustrated page views:	Indicates the number of frustrated page views in this group.	Number	If the *Average page load time* of a page is over 4 times the **SLOW TRANSACTION CUTTOFF** configuration of this test (i.e., > 4 * **SLOW TRANSACTION CUTOFF**), then such a page view is considered to be a **Frustrated page view**. Ideally, the value of this measure should be 0. A value higher than that of the *Satisfied page views* measure is a cause for concern, as it implies that the experience of users to this page type has been less than satisfactory. To know which pages are contributing to this sub-par experience, use the detailed diagnosis of this measure. The detailed

Measurement	Measurement	Measurement Unit	Interpretation
			metrics will also enable you to accurately isolate what is causing the frustrated page views – a problem with the front end? network? or backend?
Average front end time:	Indicates the interval between the arrival of the first byte of text response and the completion of the response page rendering by the browser for this page group.	ms	In the Figure 3.76 that depicts a typical page loading process, the *Average front end time* denotes the time from the *responseStart* event to the *loadEventEnd* . This process includes document downloading, processing, and page rendering. This time is therefore the sum of the *Average DOM ready time* and the *Average page rendering time*. If the *Average page load time* exceeds its threshold, then you may want to compare the value of this measure with that of the *Average server connection time* and *Average response available time* to zoom into the source of the slowness – is it the front end? the network? or the backend?
Average page rendering time:	Indicates the time taken to complete the download of remaining resources, including images, and to finish rendering the pages in this group in the browser.	ms	A high value of this measure indicates that the pages in this group are taking too long to be rendered. This can adversely impact the *Average front end time*, which in turn can prolong the *Average page load time*. Ideally therefore, the value of this measure should be low.
Average DOM ready time:	Indicates the time taken by the browser to make the complete HTML document (DOM) available for JavaScript to apply rendering logic on the pages in this group.	ms	The value of this measure is the sum of the *Average DOM download time* and the *Average DOM processing time* measures. If the value of this measure is very high, then you may want to compare the *Average DOM download time* and the *Average DOM processing time* measures to figure out what is delaying DOM building – downloading? Or processing?

Measurement	Measurement	Measurement Unit	Interpretation
			A high value for this measure can adversely impact the *Average front end time*, which in turn can prolong the *Average page load time*. Ideally therefore, the value of this measure should be low.
Average DOM download time:	Indicates the time taken to download the complete HTML document for this page group on the browser.	ms	Higher the download time of the document, longer will be the time taken to make the document available for page rendering. As a result, the overall user experience will be affected! This is why, a low value is desired for this measure at all times.
Average DOM processing time:	Indicates the time taken by the browser to build the Document Object Model (DOM) for the pages in this group and make it available for JavaScript to apply rendering logic.	ms	An unusually high value for this measure is a clear indicator that DOM building is taking longer than normal. In consequence, page rendering will be delayed, thus adversely impacting user experience when this page group is accessed. Ideally therefore, the value of this measure should be low.
Average first byte time:	Indicates the interval between the time that a user initiates a request and the time that the browser receives the first response byte for this page group. In the context of an Ajax request, this is the interval between the Ajax request dispatch and the time that the browser receives the first response byte.	ms	Going by Figure 3.76, the *Average first byte time* is the time that elapsed between *navigationStart* and *responseStart*. The value of this measure is also the sum of *Average response available time*, *Average DNS lookup time*, and *Average TCP connection time*. This means that an abnormal increase in any of the above-mentioned time values will increase the value of this measure. If the first response byte from the target

Measurement	Measurement	Measurement Unit	Interpretation
			web site/web application is itself received slowly, it is bound to have a cascading effect on all events that follow – such as, document downloading, processing, and page rendering. Ultimately, this will impact the page load time as seen by end-users. This is why, if the *Average first byte time* violates its threshold, administrators need to instantly switch to the troubleshooting mode and rapidly isolate what is causing it – is DNS lookup taking a long time? is the network connection to the web site/web application latent? or is the web server/web application server hosting the web site slow in processing requests? By comparing the values of the *Average response available time*, *Average DNS lookup time* , and *Average TCP connection time* measures, administrators can swiftly and accurately figure out the exact reason why there was a delay in receiving the first response byte. If this comparison reveals that the *Average DNS lookup time* is the highest, it implies that domain name resolution by the DNS server is taking a long time and impacting responsiveness. If the *Average TCP connection time* is found to be the culprit, then blame the network connection for delaying the transmission of the response byte. If the *Average reponse available time* is higher than the rest, you can be rest assured that the source of the problem lies with the server hosting the web site/web application.
Average response available time:	Indicates the interval between the start of	ms	In Figure 3.76 , the *Average response available* time is the time spent between

Measurement	Measurement	Measurement Unit	Interpretation
	processing of a request on the browser for this page group to when response is received.		the *requestStart* event and *responseStart* event. Ideally, a low value is desired for this measure, as high values will certainly hurt the *Apdex scrore* of the web site/web application. The key factor that can influence the value of this measure is the request processing ability of the web server/web application server that is hosting the web site/web application being monitored. Any slowdown in the backend web server/web application server – caused by the lack of adequate processing power in or improper configuration of the backend server – can significantly delay request processing by the server. In its aftermath, the *Average response available time* will increase, leaving users with an unsatisfactory experience with the web site/web application.
Average server connection time:	Indicates the elapsed time since a user initiates a request to this page group and the start of fetching the response document from it.	ms	In Figure 3.14, the time spent between *navigationStart* and *requestStart* makes up the *Average server connection time*. This includes the time to perform DNS lookups and the time to establish a TCP connection with the server. In other words, the value of this measure is nothing but the sum of the *Average DNS lookup time* and the *Average TCP connection time* measures. Ideally, the value of this measure should be low. A very high value will often end up delaying page loading and degrading the quality of the web site service. In the event that the server connection time is high therefore, simply compare the

Measurement	Measurement	Measurement Unit	Interpretation
			values of the *Average DNS lookup time* and *Average TCP connection time* measures to know to what this delay can be attributed – a delay in domain name resolution? Or a poor network connection to the server?
Average DNS lookup time:	Indicates the time taken by the browser to perform the domain lookup for connecting to this page group.	ms	A high value for this measure will not only affect DNS lookup, but will also impact the *Average server connection time* and *Average page load time*. This naturally will have a disastrous effect on user experience.
Average TCP connection time:	Indicates the time taken by the browser to establish a TCP connection with the server for accessing this page group.	ms	A bad network connection between the browser client and the server can delay TCP connections to the server As a result, the *Average server connection time* too will increase, thus impacting page load time and overall user experience with the web site/web application.

3.6.6 The Location Tests

One of the key reasons why enterprises deploy their critical applications on the cloud is to enable end-users to access that application from anywhere and at any time! Such applications naturally will be used by users from different parts of the world. In such a situation, the onus of ensuring a consistent experience for all users, regardless of their geographic location, rests with the administrator. Towards this end, administrators should first understand how user experience varies with geography, identify which location is seeing more problems than the rest, and accurately isolate the reason for the problems - is it a slow front end? is it a bad network connection? or a busy backend? - so that the bottleneck can be cleared before users notice.

To help administrators achieve this, eG Enterprise provides three different location-centric perspectives to user experience. These are as follows:

- Region
- Country
- City

The eG agent runs three different tests to provide these perspectives. These tests have been discussed in the sub-sections that follow.

3.6.6.1 Regions Test

The Regions test provides an aggregate of the experience of users from different geographic regions. If multiple users to a web site/web application are seeing slow page loads or error responses at around the same time,administrators can use the **Regions** test to figure out if the problem is region-specific or not; if so, the test further pinpoints which exact region is impacted and why is the experience of users from that region poor - is the front end used by the users in this region inefficient? Is the WAN connection between the clients in this region and the backend server hosting the web site/web application latent? or is the problem with the backend? Detailed diagnostics provided by the test also point to the specific pages that are slow or threw JavScript errors.

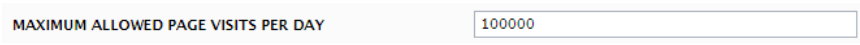
Target of the test : A web site/web application managed as a *Real User Monitor*

Agent deploying the test : A remote agent

Outputs of the test : One set of results for each browser used for accessing the web site/web application being monitored

Test parameters:

TEST PERIOD	How often should the test be executed
PROXYHOST	If the eG agent communicates with the RUM collector via a proxy, then specify the IP address/fully-qualified host name of the proxy server here. By default, this is set to *none*, indicating that the eG agent does not communicate with the collector via a proxy.
PROXYPORT	If the eG agent communicates with the RUM collector via a proxy, then specify the port at which the proxy server listens for requests from the eG agent. By default, this is set to *none*, indicating that the eG agent does not communicate with the collector via a proxy.
PROXYUSERNAME and **PROXYPASSWORD**	If the eG agent communicates with the RUM collector via a proxy server, and if proxy server requires authentication, then specify valid credentials for authentication against **PROXYUSERNAME** and **PROXYPASSWORD**. If no proxy server is used or if the proxy server used does not require authentication, then set the **PROXYUSERNAME** and **PROXYPASSWORD** to *none*.
CONFIRM PASSWORD	Confirm the **PROXYPASSWORD** by retyping it here. **Note:** If you **Reconfigure** the test later to change the values of the **PROXYUSERNAME**, **PROXYPASSWORD**, and **CONFIRM PASSWORD** parameters, then such changes will be effected only if the eG remote agent monitoring the Real User Monitor component is restarted.
DO YOU WANT TO LIMIT THE PAGE VIEWS?	By default, the eG RUM monitors all requests to a managed web site. This is why, this flag is set to **No** by default. However, in case of web sites that receive thousands of hits every day, monitoring each page view may add significantly to the overhead of the

	eG agent and may also increase the size of the eG database considerably. To reduce the strain on both the eG agent and the eG backend, you may want to restrict the monitoring scope of this test to a few page visits. To achieve this, first set this flag to **Yes**. This will invoke the option depicted by Figure 3.15.  Figure 3.82: Configuring the number of allowed page visits By default, the **MAXIMUM ALLOWED PAGE VISITS PER DAY** is set to *100000*. This implies that the test will consider only the first 100000 requests in a day for monitoring. All page visits beyond 100000 will by default be excluded from the test's monitoring purview. You can increase or decrease this limit, if you so need.
URL SEGMENTS TO BE USED AS GROUPED URL	**This parameter is applicable to the Page Groups test alone. The Page Groups test groups URLs based on the URL segments configured for monitoring and reports aggregated response time metrics for every group.** Using this parameter, you can specify a comma-separated list of URL segment numbers based on which the pages are to be grouped. URL segments are the parts of a URL (after the base URL) or path delimited by slashes. So if you had the URL: http://www.eazykart.com/web/shopping/login.jsp, then http://www.eazykart.com will be the base URL or domain, */web* will be the first URL segment, */shopping* will be the second URL segment, and */login.jsp* will be the third URL segment. By default, this parameter is set to *1,2*. This default setting, when applied to the sample URL provided above, implies that the eG agent will aggregate request and response time metrics to all instrumented web pages (i.e., web pages with the code snippet) under the URL */web/shopping*. Here, */web* corresponds to the specification 1 (URL segment 1) and */shopping* corresponds to the specification 2 (URL segment 2) in the default value *1,2*. This in turn means that, if the web site consists of pages such as http://www.eazykart.com/web/shopping/products.jsp , http://www.eazykart.com/web/shopping/products/travel/bags.jsp , http://www.eazykart.com/web/shopping/payment.jsp, etc., then the eG agent will track the requests to and responses from all these web pages, aggregate the results, and present the aggregated metrics for the descriptor */web/shopping*. This way, the test will create different page groups based on each of the second-level URL segments in the managed web site – eg., */web/movies* , */web/travel* , */web/reservations* , */partner/contacts* etc. – and will report aggregated metrics for each group so created. If you want, you can override the default setting by providing different URL segment numbers here. For instance, your specification can be just *2*. In this case, for the URL http://www.eazykart.com/web/shopping/login.jsp, the test will report metrics for the descriptor */shopping*. You can even set this parameter to 1,3. For a web site that contains URLs such as http://www.eazykart.com/web/shopping/products.jsp , http://www.eazykart.com/web/shopping/products/travel/bags.jsp , and

	http://www.eazykart.com/web/shopping/payment.jsp, this specification will result in the following descriptors: */web/products*, */web/products.jsp*, and */web/payment.jsp*.
URL PATTERNS TO BE IGNORED FROM MONITORING	By default, this test does not track requests to the following URL patterns: **.js,*.css,*.jpeg,*.jpg,*.png*. If required, you can remove one/more patterns from this default list, so that such patterns are monitored, or can append more patterns to this list in order to exclude them from monitoring. For instance, to additionally ignore URLs that end with *.gif* and *.bmp* when monitoring, you need to alter the default specification as follows: **.js,*.css,*.jpeg,*.jpg,*.png,*.gif,*.bmp* **Note:** The URL patterns configured here are not just used by the eG agent to filter out unimportant performance data during metrics collection; these patterns are also used by the RUM collector to determine which beacons should be accepted and which ones need to be discarded. Typically, the very first time this test runs and polls the RUM collector for metrics, the eG agent executing this test searches the performance records stored in the collector for data that pertains to the URL patterns configured for exclusion. If such data is found, the agent then ignores that data during metrics collection. This means that during the very first test execution, URL filtering is performed only by the eG agent. During this time, the RUM collector downloads the URL patterns configured against this parameter from the eG agent. Armed with this information, the collector then scans all beacons that browsers send subsequently, and determines if there are any beacons for the excluded URL patterns. If such beacons are found, the collector discards them instantly. Filtering URLs at the collector-level significantly reduces the load on the collector and conserves storage space on the collector; it also minimizes the workload of the eG agent. By additionally filtering URLs at the agent-level, eG makes sure that even if beacons pertaining to excluded URL patterns find their way into the RUM collector, they are captured and siphoned out by the eG agent.
JAVASCRIPT ERRORS TO BE IGNORED	By default, this test alerts administrators to all Javascript errors that occur in the monitored web site/web application. This is why, this parameter is set to *none* by default. Sometimes however, administrators may not want to be notified when certain types of Javascript errors occur – this could be because such errors are harmless or are a normal occurrence in their environment. In such circumstances, you can instruct the eG agent to ignore these errors when monitoring. For this, specify the Java script error message to be ignored in the **JAVASCRIPT ERRORS TO BE IGNORED** text box, in the following format: *<Javascript eror message>:<URL of the page/file where the message originated>*. For instance, say that the *login.html* page in your web site runs a few Java scripts that throw *Object expected* errors, which you want the eG agent to ignore. In this case, your error specification can be as follows: *Object expected:http://www.eazykart.com/web/login.html*. Alternatively, you can provide only that text string with which the error message begins in your specification – eg., *Object:http://www.eazykart.com/web/login.html*. Moreover, instead of the complete

	URL, you can specify just the name of the HTML/jsp/aspx page in which the error is to be ignored – example: *Object:login.html*. Sometimes, the individual web pages in your web site may not run any Java script directly. Instead, these web pages may include links to Java script files that will run the Java script and return the output to the web pages. If you want the eG agent to ignore certain errors thrown by such a Javascript file, then your error pattern specification should include the URL of the Javascript file and not the web page that points to it. This is because, in this case, the file is where the error message originates. For instance, in the same example above, if the *login.html* page points to a *validate.js* file, and you want to ignore the *Object expected errors* that this JS file throws, your error pattern specification will either be, *Object expected:validate.js*, or *Object:validate.js*. Multiple error message-URL combinations can also be provided as a comma-separated list. The format of your specification will be: <Javascript error message 1>:<Originating URL 1>,<Javascript error message 2>:<Originating URL 2>,. . . For example, to ignore the *Object expected* and *Uncaught TypeError* errors in the *login.html* page, use the following specification: *Object:login.html,Uncaught:login.html* Likewise, to ignore the *Object expected* error in the *login.html* page and the *Uncaught TypeError* in the *validate.js* file, your specification will be: *Object:login.html,Uncaught:validate.js* If you want to ignore the *Uncaught TypeError* across all the pages of the web site, your specification will be as follows: *Uncaught TypeError:All* **Note:** When specifying the <*Javascript error message*> to be ignored, take care of the following: • The error message should not contain any special characters – in particular, the ':' (the colon) and the ',' (the comma) characters should be avoided. • The case of the actual error message and the one specified as part of your error specification should match. This is because, the eG agent performs *case-sensitive* pattern matching.
MAXIMUM HEALTHY TRANSACTIONS IN DD	By default, this parameter is set to *5* indicating that the detailed diagnosis of this test will collect and display metrics related to the top-5 normal/healthy page views, in terms of the *Average page load time*. To identify the top 5, the eG agent sorts all healthy transactions in the ascending order of their *Average page load time*, and picks the first 5 transactions from this sorted list for display in the **DETAILED DIAGNOSIS** page and for storing in the eG database.

	You can however, increase or decrease this value depending upon how many healthy transactions you want to see in your detailed diagnosis, and how well-tuned or well-sized the eG database is. If you do not want the detailed diagnosis to include any healthy transaction, set the value of this parameter to *0*. To view all healthy transactions, set the value of this parameter to *all*. Before setting the value of this parameter to *all*, make sure that you have a well-sized eG database in place, as this setting will store details of every transaction that registers an *Avg page load time* value that is lower than the **SLOW TRANSACTION CUTOFF** specification. On a good day, this can result in numerous transactions, and can consume considerable space in the eG database.
MAXIMUM SLOW TRANSACTIONS IN DD	By default, this parameter is set to *5* indicating that the detailed diagnosis of this test will display metrics related to all the top-5 slow transactions, in terms of the *Average page load time*. To identify the top 5, the eG agent sorts all slow transactions in the descending order of their *Average page load time*, and picks the first 5 transactions from this sorted list for display in the **DETAILED DIAGNOSIS** page and for storing in the eG database. You can however, increase or decrease this value depending upon how many slow transactions you want to see in your detailed diagnosis, and how well-tuned or well-sized the eG database is. If you do not want the detailed diagnosis to include any slow transaction, set the value of this parameter to *0*. To view all slow transactions, set the value of this parameter to *all*. Before setting the value of this parameter to *all*, make sure that you have a well-sized eG database in place, as this setting will store details of every transaction that registers an *Avg page load time* value that is higher than the **SLOW TRANSACTION CUTOFF** specification. On a bad day, this can result in numerous transactions, and can consume considerable space in the eG database.
MAXIMUM ERROR TRANSACTIONS IN DD	By default, this parameter is set to *5*, indicating that the detailed diagnosis of this test will display metrics related to the top-5 transactions that encountered JavaScript errors, based on when those errors occurred. To identify the top 5, the eG agent sorts all error transactions in the descending order of the date/time at which the errors occurred, and picks the first 5 transactions from this sorted list for display in the **DETAILED DIAGNOSIS** page and for storing in the eG database. You can however, increase or decrease this value depending upon how many error transactions you want to see in your detailed diagnosis, and how well-tuned or well-sized the eG database is. If you do not want the detailed diagnosis to include any error transaction, set the value of this parameter to *0*. To view all error transactions, set the value of this parameter to *all*. Before setting the value of this parameter to *all*, make sure that you have a well-sized eG database in place, as this setting will store details of every transaction that encounters a JavaScript error. On a bad day, this can result in numerous transactions, and can consume considerable space in the eG database.

SLOW TRANSACTION CUTOFF (MS)	This test reports the count of slow page views and also pinpoints the pages that are slow. To determine whether/not a page is slow, this test uses the **SLOW TRANSACTION CUTOFF** parameter. By default, this parameter is set to *4000 millisecs* (i.e., 4 seconds). This means that, if a page takes more than 4 seconds to load, this test will consider that page as a slow page by default. You can increase or decrease this slow transaction cutoff according to what is 'slow' and what is 'normal' in your environment. **Note:** The default value of this parameter is the same as the default *Maximum threshold* setting of the *Avg page load time* measure – i.e., both are set to *4000 millisecs* by default. While the former helps eG to distinguish between slow and healthy page views for the purpose of providing detailed diagnosis, the latter tells eG when to generate an alarm on *Avg page load time*. For best results, it is recommended that both these settings are configured with the same value at all times. Therefore, if you change the value of one of these configurations, then make sure you update the value of the other as well. For instance, if the **SLOW TRASACTION CUTOFF** is changed to *6000 millisecs*, change the *Maximum Threshold* of the *Avg page load time* measure to *6000* millsecs as well.
SEND ZERO VALUES WHEN THERE IS NO TRAFFIC	By default, this flag is set to **No**. This implies that, if there is no traffic to a monitored web site/web application – i.e., if all measures of this test return only the value *0* - then the eG agent will not report these metrics to the eG manager. This also means that, by default, users to the eG monitoring console will not know that there is no traffic to the web site/web application. You can however, ensure that users to the eG monitoring console are informed of the absence of any user activity on the web site/web application. For this, set this flag to **Yes**. If this is done, then the eG agent will report all the metrics of this test to the eG manager, despite the fact that their value is *0*. These zero values will clearly indicate to users that there no traffic to the monitored web site/web application.
PAGELOADTIME CUTOFF (MS)	eG RUM relies on the Navigation API to track page views and capture page load time. Sometimes, the Navigation API can incorrectly report abnormally high values for page load time. To ensure that the eG agent ignores beacons with such page load time values, you can configure in this text box, the maximum value for page load time. By default, this is set to 3600000 (ms). This implies that by default, the eG agent will disregard all beacons that carry page load time values higher than 3600000 milliseconds. You can change this value to suit your environment.
PAGE TYPES TO BE INCLUDED IN DASHBOARD	By default, the eG RUM Dashboard displays details of base page views only. You can optionally include Ajax and/or iFrame views as well in the dashboard, by selecting the relevant options from this list box.
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.
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.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Page views:	Indicates the total number of times pages were viewed by users from this region.	Number	This is a good measure of the traffic from a specific region. Sudden, but significant spikes in the page view count could be a cause for concern, as it could be owing to a malicious virus attack or an unscrupulous attempt to hack your web site/web application. **Note:** An abnormally high value for this measure may not always be a cause for concern; nor would it always indicate a genuine increase in traffic to the web site/web application. If the eG agent monitoring the *Real User Monitor* component is stopped for sometime (say, for maintenance purposes) and then started, or if the eG agent-collector connection breaks and is restored after a while, the eG agent will pull all the metrics that the collector stored locally during the period of its absence, cumulate them, and then display the cumulated values in the eG monitoring console as metrics that pertain

Measurement	Description	Measurement Unit	Interpretation
			to the current measurement period. In reality, these metrics pertain to the entire time period when the eG agent was unavailable. Because of this, the *Page views* measure may indicate a sudden and significant surge in traffic.
Apdex score:	Indicates the apdex score of the web site/web application based on the experience of users from this region.	Number	**Apdex (Application Performance Index)** is an open standard developed by an alliance of companies. It defines a standard method for reporting and comparing the performance of software applications in computing. Its purpose is to convert measurements into insights about user satisfaction, by specifying a uniform way to analyze and report on the degree to which measured performance meets user expectations. The Apdex method converts many measurements into one number on a uniform scale of 0-to-1 (0 = no users satisfied, 1 = all users satisfied). The resulting Apdex score is a numerical measure of user satisfaction with the performance of enterprise applications. This metric can be used to report on any source of end- user performance measurements for which a performance objective has been defined. The Apdex formula is: $Apdex_t$ = *(Satisfied Count + Tolerating Count / 2) / Total Samples* This is nothing but the number of satisfied samples plus half of the tolerating samples plus none of the frustrated samples, divided by all the samples. A score of 1.0 means all responses were satisfactory. A score of 0.0 means none

Measurement	Description	Measurement Unit	Interpretation
			of the responses were satisfactory. Tolerating responses half satisfy a user. For example, if all responses are tolerating, then the Apdex score would be 0.50. Ideally therefore, the value of this measure should be *1.0*. A value less than *1.0* indicates that the experience of users from this region has been less than satisfactory.
Average page load time:	Indicates the average time that pages accessed by users from this region took to load completely on the browser.	ms	This is the average interval between the time that a user initiates a request and the completion of the page load of the response in the user's browser. In the context of an Ajax request, it ends when the response has been completely processed. By comparing the value of this measure across regions, you will be able to tell if the page load time is significantly higher for any one region – this is the first sign that the problem could be regional. Next, check the detailed diagnosis of this measure for this region to find out which pages are slow. Then, look up the detailed measures for the other regions to ascertain whether users from those regions also experienced slowness when accessing the same pages. If not, it is a sure sign that the problem is specific to the region. So, proceed to compare the values of the *Average front end time*, *Average server connection time*, and *Average response available time* measures for that region, to know why the users in that region are seeing slowness – is it the front end? network? or the backend?

Measurement	Description	Measurement Unit	Interpretation
			If the *Average front end time* is the maximum, then the problem is the front end used by the users in that region. If the *Average server connection time* is the highest, then the network connection between the user's browser and the web site/web application is the one contributing to the slowness. A very high *Average response available time* on the other hand, reveals that problem may have nothing to do with the geographic region, but with the server that is hosting the web site/web application.
Unique user session:	Indicates the number of distinct users who are currently accessing the web site/web application from this region.	Number	
Page views per minute:	Indicates the number of times the pages were viewed per minute by users from this region.	Number	An unusually high value for this measure may require investigation.
Normal page view percentage:	Indicates the percentage of page views that delivered a satisfactory experience to users from this region.	Percent	The value of this measure indicates the percentage of page views in which users from this region have neither experienced any slowness, nor encountered any Javascript errors. Ideally, the value of this measure should be 100%. A value that is slightly less than 100% indicates that the user experience has not been up to the mark. A value less than 50% is indicative of a serious problem, where most of the page views are either slow or have encountered Javascript errors. Under such circumstances, to know what exactly is

Measurement	Description	Measurement Unit	Interpretation
			affecting the experience of users, compare the value of the *Slow page view percentage* with that of the *Javasript error page view percentage* for that browser. This will reveal the reason for the poor user experience – slow pages? or Javascript errors? If slow pages are the problem, use the detailed diagnosis of the *Slow page views* measure to know which pages are slow and where these pages are losing time – in the front end? network? or backend?. If JavaScript errors are the problem, use the detailed diagnosis of the *JavaScript error view percentage* measure to know what errors occurred in which pages.
Slow page view percentage:	Indicates the percentage of pages that were slow in loading when accessed by users in this region.	Percent	Ideally, the value of this measure should be 0. A value over 50% implies that you are in a spot of bother, with over half of the page views being slow. Use the detailed diagnosis of the *Slow page views* measure to identify the slow pages and isolate the root-cause of the slowness – is it the front end? the network? or the backend?
JavaScript error view percentage:	Indicates the percentage of page views that have encountered JavaScript errors.	Percent	Ideally, the value of this measure should be 0. A value over 50% implies that you are in a spot of bother, with over half of the page views experiencing JavaScript errors.
Slow page views (Tolerating & Frustrated):	Indicates the number of slow page views from this region.	Number	A page view is considered to be slow when the average time taken to load that page exceeds the **SLOW TRANSACTION CUTOFF** configured for this test. Ideally, a page should load quickly. The

Measurement	Description	Measurement Unit	Interpretation
			value 0 is hence desired for this measure. If the value of this measure is high, it indicates that users from this region frequently experienced slowness when accessing pages in the web site/web application. To know which page views are slow and why, use the detailed diagnosis of this measure.
JavaScript error page views:	Indicates the number of times JavaScript errors occurred when accessing pages from this region.	Number	Ideally, the value of this measure should be 0. A high value indicates that many JavaScript errors are occurring when accessing pages from this region. Use the detailed diagnosis of this measure to identify the error pages and to know what Javascript error has occurred in which page. This will greatly aid troubleshooting!
Satisfied page views:	Indicates the number of times pages were viewed by users in this region without any slowness.	Number	A page view is considered to be slow when the average time taken to load that page exceeds the **SLOW TRANSACTION CUTOFF** configured for this test. If this **SLOW TRANSACTION CUTOFF** is not exceeded, then the page view is deemed to be 'satisfactory'. To know which page views are satisfactory, use the detailed diagnosis of this measure. Ideally, the value of this measure should be the same as that of the *Page views* measure. If not, then it indicates that one/more page views are slow – i.e., have violated the **SLOW TRANSACTION CUTOFF**. If the value of this measure is much lesser than the value of the *Tolerating page views* and the *Frustrated page views*, it is a clear indicator that the experience of users in this region has

Measurement	Description	Measurement Unit	Interpretation
			been below-par. In such a case, use the detailed diagnosis of the *Tolerating page views* and *Frustrated page views* measures to know which pages are slow and why.
Tolerating page views:	Indicates the number of tolerating page views experienced by users in this region.	Number	If the *Average page load time* of a page exceeds the **SLOW TRANSACTION CUTTOFF** configuration of this test, but is less than 4 times the **SLOW TRANSACTION CUTOFF** (i.e., < 4 * **SLOW TRANSACTION CUTOFF**), then such a page view is considered to be a **Tolerating page view**. Ideally, the value of this measure should be 0. A value higher than that of the *Satisfied page views* measure is a cause for concern, as it implies that the overall experience of the users in this region is less than satisfactory. To know which pages are contributing to this sub-par experience, use the detailed diagnosis of this measure. The detailed metrics will also enable you to accurately isolate what is causing the tolerating page views – a problem with the front end? network? or backend?
Frustrated page views:	Indicates the number of frustrated page views experienced by users in this region.	Number	If the *Average page load time* of a page is over 4 times the **SLOW TRANSACTION CUTTOFF** configuration of this test (i.e., > 4 * **SLOW TRANSACTION CUTOFF**), then such a page view is considered to be a **Frustrated page view**. Ideally, the value of this measure should be 0. A value higher than that of the *Satisfied page views* measure is a cause for concern, as it implies that the experience of users in this region has

Measurement	Description	Measurement Unit	Interpretation
			been less than satisfactory. To know which pages are contributing to this sub-par experience, use the detailed diagnosis of this measure. The detailed metrics will also enable you to accurately isolate what is causing the frustrated page views – a problem with the front end? network? or backend?
Average front end time:	Indicates the interval between the arrival of the first byte of text response and the completion of the response page rendering by the browser used by users in this region.	ms	In the Figure 3.76 that depicts a typical page loading process, the *Average front end time* denotes the time from the *responseStart* event to the *loadEventEnd* . This process includes document downloading, processing, and page rendering. This time is therefore the sum of the *Average DOM ready time* and the *Average page rendering time*. If the *Average page load time* exceeds its threshold, then you may want to compare the value of this measure with that of the *Average server connection time* and *Average response available time* to zoom into the source of the slowness – is it the front end? the network? or the backend?
Average page rendering time:	Indicates the time taken to complete the download of remaining resources, including images, and to finish rendering the pages in the browser for users in this region.	ms	A high value of this measure indicates that the pages are taking too long to be rendered. This can adversely impact the *Average front end time*, which in turn can prolong the *Average page load time* . Ideally therefore, the value of this measure should be low.
Average DOM ready time:	Indicates the time taken by the browser to make the complete HTML document (DOM) available for JavaScript to apply	ms	The value of this measure is the sum of the *Average DOM download time* and the *Average DOM processing time* measures. If the value of this measure is very high, then you may want to compare

Measurement	Description	Measurement Unit	Interpretation
	rendering logic on the pages accessed by users in this region.		the *Average DOM download time* and the *Average DOM processing time* measures to figure out what is delaying DOM building – downloading? Or processing? A high value for this measure can adversely impact the *Average front end time*, which in turn can prolong the *Average page load time*. Ideally therefore, the value of this measure should be low.
Average DOM download time:	Indicates the time taken to download the complete HTML document for requests received from users in this region.	ms	Higher the download time of the document, longer will be the time taken to make the document available for page rendering. As a result, the overall user experience will be affected! This is why, a low value is desired for this measure at all times.
Average DOM processing time:	Indicates the time taken by the browser to build the Document Object Model (DOM) for the pages requested by the users in this region and make it available for JavaScript to apply rendering logic.	ms	An unusually high value for this measure is a clear indicator that DOM building is taking longer than normal. In consequence, page rendering will be delayed, thus adversely impacting user experience when users in this region are accessing the web site/web application. Ideally therefore, the value of this measure should be low.
Average first byte time:	Indicates the interval between the time that a user in this region initiates a request and the time that the browser receives the first response byte. In the context of an Ajax request, this is the interval between the Ajax request dispatch and the time that the browser receives the first response byte.	ms	Going by Figure 3.76 above, the *Average first byte time* is the time that elapsed between *navigationStart* and *responseStart*. The value of this measure is also the sum of *Average response available time*, *Average DNS lookup time*, and *Average TCP connection time*. This means that an abnormal increase in any of the above-mentioned time values will increase the value of this measure.

Measurement	Description	Measurement Unit	Interpretation
			If the first response byte from the target web site/web application is itself received slowly, it is bound to have a cascading effect on all events that follow – such as, document downloading, processing, and page rendering. Ultimately, this will impact the page load time as seen by users in this region. This is why, if the *Average first byte time* violates its threshold, administrators need to instantly switch to the troubleshooting mode and rapidly isolate what is causing it – is DNS lookup taking a long time? is the network connection to the web site/web application latent? or is the web server/web application server hosting the web site slow in processing requests? By comparing the values of the *Average response available time*, *Average DNS lookup time* , and *Average TCP connection time* measures, administrators can swiftly and accurately figure out the exact reason why there was a delay in receiving the first response byte. If this comparison reveals that the *Average DNS lookup time* is the highest, it implies that domain name resolution by the DNS server is taking a long time and impacting responsiveness. If the *Average TCP connection time* is found to be the culprit, then blame the network connection for delaying the transmission of the response byte. If the *Average reponse available time* is higher than the rest, you can be rest assured that the source of the problem lies with the server hosting the web site/web application.
Average response	Indicates the interval	ms	In Figure 3.76 , the *Average response*

Measurement	Description	Measurement Unit	Interpretation
available time:	between the start of processing of a request from a user in this region to when response is received.		*available* time is the time spent between the *requestStart* event and *responseStart* event. Ideally, a low value is desired for this measure, as high values will certainly hurt the *Apdex scrore* of the web site/web application. The key factor that can influence the value of this measure is the request processing ability of the web server/web application server that is hosting the web site/web application being monitored. Any slowdown in the backend web server/web application server - caused by the lack of adequate processing power in or improper configuration of the backend server – can significantly delay request processing by the server. In its aftermath, the *Average response available time* will increase, leaving users with an unsatisfactory experience with the web site/web application.
Average server connection time:	Indicates the elapsed time since a user in this region initiates a request to the web site/web application and the start of fetching the response document from it.	ms	In Figure 3.76, the time spent between *navigationStart* and *requestStart* makes up the *Average server connection time*. This includes the time to perform DNS lookups and the time to establish a TCP connection with the server. In other words, the value of this measure is nothing but the sum of the *Average DNS lookup time* and the *Average TCP connection time* measures. Ideally, the value of this measure should be low. A very high value will often end up delaying page loading and degrading the quality of the web site service. In the event that the server connection time is

Measurement	Description	Measurement Unit	Interpretation
			high therefore, simply compare the values of the *Average DNS lookup time* and *Average TCP connection time* measures to know to what this delay can be attributed – a delay in domain name resolution? Or a poor network connection to the server?
Average DNS lookup time:	Indicates the time taken by the browser used by a user in this region to perform the domain lookup for connecting to the web site/web application.	ms	A high value for this measure will not only affect DNS lookup, but will also impact the *Average server connection time* and *Average page load time*. This naturally will have a disastrous effect on user experience.
Average TCP connection time:	Indicates the time taken by the browser used by a user in this region to establish a TCP connection with the server.	ms	A bad network connection between the browser client and the server can delay TCP connections to the server As a result, the *Average server connection time* too will increase, thus impacting page load time and overall user experience with the web site/web application.

3.6.6.2 Countries Test

The **Countries** test provides an aggregate of the experience of users from different countries. If you have a web site/web application that is accessed by users from different countries, you can use this test to know:

- Which countries your users are coming from;
- Which are the countries in which your web site/web application is very popular
- How is the experience of users from each of these countries.

Moreover, if multiple users to a web site/web application are seeing slow page loads or error responses at around the same time, administrators can use the **Countries** test to figure out if the problem is specific to a particular country; if so, the test further pinpoints which exact country is impacted and why is the experience of users from that country poor – is the front end used by the users in this country inefficient? Is the WAN connection between the browser clients in this country and the backend server hosting the web site/web

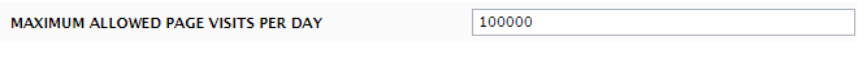
application latent? or is the problem with the backend? Detailed diagnostics provided by the test also point to the specific pages that are slow or have encountered JavaScript errors.

Target of the test : A web site/web application managed as a *Real User Monitor*

Agent deploying the test : A remote agent

Outputs of the test : One set of results for each browser used for accessing the web site/web application being monitored

Test parameters:

TEST PERIOD	How often should the test be executed
PROXYHOST	If the eG agent communicates with the RUM collector via a proxy, then specify the IP address/fully-qualified host name of the proxy server here. By default, this is set to *none*, indicating that the eG agent does not communicate with the collector via a proxy.
PROXYPORT	If the eG agent communicates with the RUM collector via a proxy, then specify the port at which the proxy server listens for requests from the eG agent. By default, this is set to *none*, indicating that the eG agent does not communicate with the collector via a proxy.
PROXYUSERNAME and **PROXYPASSWORD**	If the eG agent communicates with the RUM collector via a proxy server, and if proxy server requires authentication, then specify valid credentials for authentication against **PROXYUSERNAME** and **PROXYPASSWORD**. If no proxy server is used or if the proxy server used does not require authentication, then set the **PROXYUSERNAME** and **PROXYPASSWORD** to *none*.
CONFIRM PASSWORD	Confirm the **PROXYPASSWORD** by retyping it here. **Note:** If you **Reconfigure** the test later to change the values of the **PROXYUSERNAME**, **PROXYPASSWORD**, and **CONFIRM PASSWORD** parameters, then such changes will be effected only if the eG remote agent monitoring the Real User Monitor component is restarted.
DO YOU WANT TO LIMIT THE PAGE VIEWS?	By default, the eG RUM monitors all requests to a managed web site. This is why, this flag is set to **No** by default. However, in case of web sites that receive thousands of hits every day, monitoring each page view may add significantly to the overhead of the eG agent and may also increase the size of the eG database considerably. To reduce the strain on both the eG agent and the eG backend, you may want to restrict the monitoring scope of this test to a few page visits. To achieve this, first set this flag to **Yes**. This will invoke the option depicted by the below figure.  Figure 3.83: Configuring the number of allowed page visits By default, the **MAXIMUM ALLOWED PAGE VISITS PER DAY** is set to *100000*. This implies that the test will consider only the first 100000 requests in a day for monitoring. All page

	visits beyond 100000 will by default be excluded from the test's monitoring purview. You can increase or decrease this limit, if you so need.
URL SEGMENTS TO BE USED AS GROUPED URL	**This parameter is applicable to the Page Groups test alone**. The **Page Groups** test groups URLs based on the URL segments configured for monitoring and reports aggregated response time metrics for every group. Using this parameter, you can specify a comma-separated list of URL segment numbers based on which the pages are to be grouped. URL segments are the parts of a URL (after the base URL) or path delimited by slashes. So if you had the URL: http://www.eazykart.com/web/shopping/login.jsp , then http://www.eazykart.com will be the base URL or domain, */web* will be the first URL segment, */shopping* will be the second URL segment, and */login.jsp* will be the third URL segment. By default, this parameter is set to *1,2*. This default setting, when applied to the sample URL provided above, implies that the eG agent will aggregate request and response time metrics to all instrumented web pages (i.e., web pages with the code snippet) under the URL */web/shopping*. Here, */web* corresponds to the specification 1 (URL segment 1) and */shopping* corresponds to the specification 2 (URL segment 2) in the default value *1,2*. This in turn means that, if the web site consists of pages such as http://www.eazykart.com/web/shopping/products.jsp , http://www.eazykart.com/web/shopping/products/travel/bags.jsp , http://www.eazykart.com/web/shopping/payment.jsp, etc., then the eG agent will track the requests to and responses from all these web pages, aggregate the results, and present the aggregated metrics for the descriptor */web/shopping*. This way, the test will create different page groups based on each of the second-level URL segments in the managed web site – eg., */web/movies* , */web/travel* , */web/reservations* , */partner/contacts* etc. – and will report aggregated metrics for each group so created. If you want, you can override the default setting by providing different URL segment numbers here. For instance, your specification can be just *2*. In this case, for the URL http://www.eazykart.com/web/shopping/login.jsp, the test will report metrics for the descriptor */shopping*. You can even set this parameter to 1,3. For a web site that contains URLs such as http://www.eazykart.com/web/shopping/products.jsp , http://www.eazykart.com/web/shopping/products/travel/bags.jsp , and http://www.eazykart.com/web/shopping/payment.jsp, this specification will result in the following descriptors: */web/products*, */web/products.jsp*, and */web/payment.jsp*.
URL PATTERNS TO BE IGNORED FROM MONITORING	By default, this test does not track requests to the following URL patterns: **.js,*.css,*.jpeg,*.jpg,*.png*. If required, you can remove one/more patterns from this default list, so that such patterns are monitored, or can append more patterns to this list in order to exclude them from monitoring. For instance, to additionally ignore URLs that end with *.gif* and *.bmp* when monitoring, you need to alter the default specification as follows: **.js,*.css,*.jpeg,*.jpg,*.png,*.gif,*.bmp* **Note:**

	The URL patterns configured here are not just used by the eG agent to filter out unimportant performance data during metrics collection; these patterns are also used by the RUM collector to determine which beacons should be accepted and which ones need to be discarded. Typically, the very first time this test runs and polls the RUM collector for metrics, the eG agent executing this test searches the performance records stored in the collector for data that pertains to the URL patterns configured for exclusion. If such data is found, the agent then ignores that data during metrics collection. This means that during the very first test execution, URL filtering is performed only by the eG agent. During this time, the RUM collector downloads the URL patterns configured against this parameter from the eG agent. Armed with this information, the collector then scans all beacons that browsers send subsequently, and determines if there are any beacons for the excluded URL patterns. If such beacons are found, the collector discards them instantly. Filtering URLs at the collector-level significantly reduces the load on the collector and conserves storage space on the collector; it also minimizes the workload of the eG agent. By additionally filtering URLs at the agent-level, eG makes sure that even if beacons pertaining to excluded URL patterns find their way into the RUM collector, they are captured and siphoned out by the eG agent.
JAVASCRIPT ERRORS TO BE IGNORED	By default, this test alerts administrators to all Javascript errors that occur in the monitored web site/web application. This is why, this parameter is set to *none* by default. Sometimes however, administrators may not want to be notified when certain types of Javascript errors occur – this could be because such errors are harmless or are a normal occurrence in their environment. In such circumstances, you can instruct the eG agent to ignore these errors when monitoring. For this, specify the Java script error message to be ignored in the **JAVASCRIPT ERRORS TO BE IGNORED** text box, in the following format: *<Javascript eror message>:<URL of the page/file where the message originated>*. For instance, say that the *login.html* page in your web site runs a few Java scripts that throw *Object expected* errors, which you want the eG agent to ignore. In this case, your error specification can be as follows: *Object expected:http://www.eazykart.com/web/login.html*. Alternatively, you can provide only that text string with which the error message begins in your specification – eg., *Object:http://www.eazykart.com/web/login.html*. Moreover, instead of the complete URL, you can specify just the name of the HTML/jsp/aspx page in which the error is to be ignored – example: *Object:login.html*. Sometimes, the individual web pages in your web site may not run any Java script directly. Instead, these web pages may include links to Java script files that will run the Java script and return the output to the web pages. If you want the eG agent to ignore certain errors thrown by such a Javascript file, then your error pattern specification should include the URL of the Javascript file and not the web page that points to it. This is because, in this case, the file is where the error message originates. For instance, in the same example above, if the *login.html* page points to a *validate.js* file, and you want to ignore the *Object expected errors* that this JS file throws, your error pattern

	specification will either be, *Object expected:validate.js*, or *Object:validate.js*. Multiple error message-URL combinations can also be provided as a comma-separated list. The format of your specification will be: *<Javascript error message 1>:<Originating URL 1>,<Javascript error message 2>:<Originating URL 2>,. . .* For example, to ignore the *Object expected* and *Uncaught TypeError* errors in the *login.html* page, use the following specification: *Object:login.html,Uncaught:login.html* Likewise, to ignore the *Object expected* error in the *login.html* page and the *Uncaught TypeError* in the *validate.js* file, your specification will be: *Object:login.html,Uncaught:validate.js* If you want to ignore the *Uncaught TypeError* across all the pages of the web site, your specification will be as follows: *Uncaught TypeError:All* **Note:** When specifying the *<Javascript error message>* to be ignored, take care of the following: • The error message should not contain any special characters – in particular, the ':' (the colon) and the ',' (the comma) characters should be avoided. • The case of the actual error message and the one specified as part of your error specification should match. This is because, the eG agent performs *case-sensitive* pattern matching.
MAXIMUM HEALTHY TRANSACTIONS IN DD	By default, this parameter is set to *5* indicating that the detailed diagnosis of this test will collect and display metrics related to the top-5 normal/healthy page views, in terms of the *Average page load time*. To identify the top 5, the eG agent sorts all healthy transactions in the ascending order of their *Average page load time*, and picks the first 5 transactions from this sorted list for display in the **DETAILED DIAGNOSIS** page and for storing in the eG database. You can however, increase or decrease this value depending upon how many healthy transactions you want to see in your detailed diagnosis, and how well-tuned or well-sized the eG database is. If you do not want the detailed diagnosis to include any healthy transaction, set the value of this parameter to *0*. To view all healthy transactions, set the value of this parameter to *all*. Before setting the value of this parameter to *all*, make sure that you have a well-sized eG database in place, as this setting will store details of every transaction that registers an *Avg page load time* value that is lower than the **SLOW TRANSACTION CUTOFF** specification. On a good day, this can result in numerous transactions, and can consume considerable space in the eG database.

MAXIMUM SLOW TRANSACTIONS IN DD	By default, this parameter is set to *5* indicating that the detailed diagnosis of this test will display metrics related to all the top-5 slow transactions, in terms of the *Average page load time*. To identify the top 5, the eG agent sorts all slow transactions in the descending order of their *Average page load time*, and picks the first 5 transactions from this sorted list for display in the **DETAILED DIAGNOSIS** page and for storing in the eG database. You can however, increase or decrease this value depending upon how many slow transactions you want to see in your detailed diagnosis, and how well-tuned or well-sized the eG database is. If you do not want the detailed diagnosis to include any slow transaction, set the value of this parameter to *0*. To view all slow transactions, set the value of this parameter to *all*. Before setting the value of this parameter to *all*, make sure that you have a well-sized eG database in place, as this setting will store details of every transaction that registers an *Avg page load time* value that is higher than the **SLOW TRANSACTION CUTOFF** specification. On a bad day, this can result in numerous transactions, and can consume considerable space in the eG database.
MAXIMUM ERROR TRANSACTIONS IN DD	By default, this parameter is set to *5*, indicating that the detailed diagnosis of this test will display metrics related to the top-5 transactions that encountered JavaScript errors, based on when those errors occurred. To identify the top 5, the eG agent sorts all error transactions in the descending order of the date/time at which the errors occurred, and picks the first 5 transactions from this sorted list for display in the **DETAILED DIAGNOSIS** page and for storing in the eG database. You can however, increase or decrease this value depending upon how many error transactions you want to see in your detailed diagnosis, and how well-tuned or well-sized the eG database is. If you do not want the detailed diagnosis to include any error transaction, set the value of this parameter to *0*. To view all error transactions, set the value of this parameter to *all*. Before setting the value of this parameter to *all*, make sure that you have a well-sized eG database in place, as this setting will store details of every transaction that encounters a JavaScript error. On a bad day, this can result in numerous transactions, and can consume considerable space in the eG database.
SLOW TRANSACTION CUTOFF (MS)	This test reports the count of slow page views and also pinpoints the pages that are slow. To determine whether/not a page is slow, this test uses the **SLOW TRANSACTION CUTOFF** parameter. By default, this parameter is set to *4000 millisecs* (i.e., 4 seconds). This means that, if a page takes more than 4 seconds to load, this test will consider that page as a slow page by default. You can increase or decrease this slow transaction cutoff according to what is 'slow' and what is 'normal' in your environment. **Note:** The default value of this parameter is the same as the default *Maximum threshold* setting of the *Avg page load time* measure – i.e., both are set to *4000 millisecs* by default. While the former helps eG to distinguish between slow and healthy page views

	for the purpose of providing detailed diagnosis, the latter tells eG when to generate an alarm on *Avg page load time*. For best results, it is recommended that both these settings are configured with the same value at all times. Therefore, if you change the value of one of these configurations, then make sure you update the value of the other as well. For instance, if the **SLOW TRASACTION CUTOFF** is changed to *6000 millisecs*, change the *Maximum Threshold* of the *Avg page load time* measure to *6000* millsecs as well.
SEND ZERO VALUES WHEN THERE IS NO TRAFFIC	By default, this flag is set to **No**. This implies that, if there is no traffic to a monitored web site/web application – i.e., if all measures of this test return only the value *0* - then the eG agent will not report these metrics to the eG manager. This also means that, by default, users to the eG monitoring console will not know that there is no traffic to the web site/web application. You can however, ensure that users to the eG monitoring console are informed of the absence of any user activity on the web site/web application. For this, set this flag to **Yes**. If this is done, then the eG agent will report all the metrics of this test to the eG manager, despite the fact that their value is *0*. These zero values will clearly indicate to users that there no traffic to the monitored web site/web application.
PAGELOADTIME CUTOFF (MS)	eG RUM relies on the Navigation API to track page views and capture page load time. Sometimes, the Navigation API can incorrectly report abnormally high values for page load time. To ensure that the eG agent ignores beacons with such page load time values, you can configure in this text box, the maximum value for page load time. By default, this is set to 3600000 (ms). This implies that by default, the eG agent will disregard all beacons that carry page load time values higher than 3600000 milliseconds. You can change this value to suit your environment.
PAGE TYPES TO BE INCLUDED IN DASHBOARD	By default, the eG RUM Dashboard displays details of base page views only. You can optionally include Ajax and/or iFrame views as well in the dashboard, by selecting the relevant options from this list box.
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**.
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.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Page views:	Indicates the total number of times pages were viewed by users from this country.	Number	This is a good measure of the traffic from a specific country. Sudden, but significant spikes in the page view count could be a cause for concern, as it could be owing to a malicious virus attack or an unscrupulous attempt to hack your web site/web application. **Note:** An abnormally high value for this measure may not always be a cause for concern; nor would it always indicate a genuine increase in traffic to the web site/web application. If the eG agent monitoring the *Real User Monitor* component is stopped for sometime (say, for maintenance purposes) and then started, or if the eG agent-collector connection breaks and is restored after a while, the eG agent will pull all the metrics that the collector stored locally during the period of its absence, cumulate them, and then display the cumulated values in the eG monitoring console as metrics that pertain to the current measurement period. In reality, these metrics pertain to the entire time period when the eG agent was unavailable. Because of this, the *Page*

Measurement	Description	Measurement Unit	Interpretation
			views measure may indicate a sudden and significant surge in traffic.
Apdex score:	Indicates the apdex score of the web site/web application based on the experience of users from this country.	Number	**Apdex (Application Performance Index)** is an open standard developed by an alliance of companies. It defines a standard method for reporting and comparing the performance of software applications in computing. Its purpose is to convert measurements into insights about user satisfaction, by specifying a uniform way to analyze and report on the degree to which measured performance meets user expectations. The Apdex method converts many measurements into one number on a uniform scale of 0-to-1 (0 = no users satisfied, 1 = all users satisfied). The resulting Apdex score is a numerical measure of user satisfaction with the performance of enterprise applications. This metric can be used to report on any source of end- user performance measurements for which a performance objective has been defined. The Apdex formula is: $Apdex_t$ = *(Satisfied Count + Tolerating Count / 2) / Total Samples* This is nothing but the number of satisfied samples plus half of the tolerating samples plus none of the frustrated samples, divided by all the samples. A score of 1.0 means all responses were satisfactory. A score of 0.0 means none of the responses were satisfactory. Tolerating responses half satisfy a user. For example, if all responses are tolerating, then the Apdex score would

Measurement	Description	Measurement Unit	Interpretation
			be 0.50. Ideally therefore, the value of this measure should be *1.0*. A value less than *1.0* indicates that the experience of users from this country has been less than satisfactory.
Average page load time:	Indicates the average time that pages accessed by users from this country took to load completely on the browser.	ms	This is the average interval between the time that a user initiates a request and the completion of the page load of the response in the user's browser. In the context of an Ajax request, it ends when the response has been completely processed. By comparing the value of this measure across countries, you will be able to tell if the page load time is significantly higher for any one country – this is the first sign that the problem could be country-specific. Next, check the detailed diagnosis of this measure for this country to find out which pages are slow. Then, look up the detailed measures for the other countries to ascertain whether users from those countries also experienced slowness when accessing the same pages. If not, it is a sure sign that the problem is specific to the country. So, proceed to compare the values of the *Average front end time*, *Average server connection time*, and *Average response available time* measures for that country, to know why the users in that country are seeing slowness – is it the front end? network? or the backend? If the *Average front end time* is the maximum, then the problem is the front end used by the users in that country. If

Measurement	Description	Measurement Unit	Interpretation
			the *Average server connection time* is the highest, then the network connection between the user's browser and the web site/web application is the one contributing to the slowness. A very high *Average response available time* on the other hand, reveals that problem may have nothing to do with the country, but with the server that is hosting the web site/web application.
Unique user session:	Indicates the number of distinct users who are currently accessing the web site/web application from this country.	Number	
Page views per minute:	Indicates the number of times the pages were viewed per minute by users from this country.	Number	An unusually high value for this measure may require investigation.
Normal page view percentage:	Indicates the percentage of page views that delivered a satisfactory experience to users from this country.	Percent	The value of this measure indicates the percentage of page views in which users from this country have neither experienced any slowness, nor encountered any Javascript errors. Ideally, the value of this measure should be 100%. A value that is slightly less than 100% indicates that the user experience has not been up to the mark. A value less than 50% is indicative of a serious problem, where most of the page views are either slow or have encountered Javascript errors. Under such circumstances, to know what exactly is affecting the experience of users, compare the value of the *Slow page view percentage* with that of the *Javasript error*

Measurement	Description	Measurement Unit	Interpretation
			page view percentage for that browser. This will reveal the reason for the poor user experience – slow pages? or Javascript errors? If slow pages are the problem, use the detailed diagnosis of the *Slow page views* measure to know which pages are slow and where these pages are losing time – in the front end? network? or backend?. If JavaScript errors are the problem, use the detailed diagnosis of the *JavaScript error view percentage* measure to know what errors occurred in which pages.
Slow page view percentage:	Indicates the percentage of pages that were slow in loading when accessed by users in this country.	Percent	Ideally, the value of this measure should be 0. A value over 50% implies that you are in a spot of bother, with over half of the page views being slow. Use the detailed diagnosis of the *Slow page views* measure to identify the slow pages and isolate the root-cause of the slowness – is it the front end? the network? or the backend?
JavaScript error view percentage:	Indicates the percentage of page views that have encountered JavaScript errors.	Percent	Ideally, the value of this measure should be 0. A value over 50% implies that you are in a spot of bother, with over half of the page views experiencing JavaScript errors.
Slow page views (Tolerating & Frustrated):	Indicates the number of page views that were slow when accessed from this country.	Number	A page view is considered to be slow when the average time taken to load that page exceeds the **SLOW TRANSACTION CUTOFF** configured for this test. Ideally, a page should load quickly. The value 0 is hence desired for this measure. If the value of this measure is high, it indicates that users from this country

Measurement	Description	Measurement Unit	Interpretation
			frequently experienced slowness when accessing pages in the web site/web application. To know which page views are slow and why, use the detailed diagnosis of this measure.
JavaScript error page views:	Indicates the number of times JavaScript errors occurred when accessing pages from this country.	Number	Ideally, the value of this measure should be 0. A high value indicates that many JavaScript errors are occurring when accessing pages in the web site/web application. Use the detailed diagnosis of this measure to identify the error pages and to know what Javascript error has occurred in which page. This will greatly aid troubleshooting!
Satisfied page views:	Indicates the number of times pages were viewed by users in this country without any slowness.	Number	A page view is considered to be slow when the average time taken to load that page exceeds the **SLOW TRANSACTION CUTOFF** configured for this test. If this **SLOW TRANSACTION CUTOFF** is not exceeded, then the page view is deemed to be 'satisfactory'. To know which page views are satisfactory, use the detailed diagnosis of this measure. Ideally, the value of this measure should be the same as that of the *Page views* measure. If not, then it indicates that one/more page views are slow – i.e., have violated the **SLOW TRANSACTION CUTOFF**. If the value of this measure is much lesser than the value of the *Tolerating page views* and the *Frustrated page views*, it is a clear indicator that the experience of users in this country has been below-par. In such a case, use the detailed diagnosis of the *Tolerating page views* and *Frustrated page views*

Measurement	Description	Measurement Unit	Interpretation
			measures to know which pages are slow and why.
Tolerating page views:	Indicates the number of tolerating page views experienced by users in this country.	Number	If the *Average page load time* of a page exceeds the **SLOW TRANSACTION CUTTOFF** configuration of this test, but is less than 4 times the **SLOW TRANSACTION CUTOFF** (i.e., < 4 * **SLOW TRANSACTION CUTOFF**), then such a page view is considered to be a **Tolerating page view**. Ideally, the value of this measure should be 0. A value higher than that of the *Satisfied page views* measure is a cause for concern, as it implies that the overall experience of the users in this country is less than satisfactory. To know which pages are contributing to this sub-par experience, use the detailed diagnosis of this measure. The detailed metrics will also enable you to accurately isolate what is causing the tolerating page views - a problem with the front end? network? or backend?
Frustrated page views:	Indicates the number of frustrated page views experienced by users in this country.	Number	If the *Average page load time* of a page is over 4 times the **SLOW TRANSACTION CUTTOFF** configuration of this test (i.e., > 4 * **SLOW TRANSACTION CUTOFF**), then such a page view is considered to be a **Frustrated page view**. Ideally, the value of this measure should be 0. A value higher than that of the *Satisfied page views* measure is a cause for concern, as it implies that the experience of users in this country has been less than satisfactory. To know which pages are contributing to this sub-par experience, use the detailed

Measurement	Description	Measurement Unit	Interpretation
			diagnosis of this measure. The detailed metrics will also enable you to accurately isolate what is causing the frustrated page views – a problem with the front end? network? or backend?
Average front end time:	Indicates the interval between the arrival of the first byte of text response and the completion of the response page rendering by the browser used by users in this country.	ms	In the Figure 3.76 that depicts a typical page loading process, the *Average front end time* denotes the time from the *responseStart* event to the *loadEventEnd* . This process includes document downloading, processing, and page rendering. This time is therefore the sum of the *Average DOM ready time* and the *Average page rendering time*. If the *Average page load time* exceeds its threshold, then you may want to compare the value of this measure with that of the *Average server connection time* and *Average response available time* to zoom into the source of the slowness – is it the front end? the network? or the backend?
Average page rendering time:	Indicates the time taken to complete the download of remaining resources, including images, and to finish rendering the pages in the browser for users in this country.	ms	A high value of this measure indicates that the pages are taking too long to be rendered. This can adversely impact the *Average front end time*, which in turn can prolong the *Average page load time* . Ideally therefore, the value of this measure should be low.
Average DOM ready time:	Indicates the time taken by the browser to make the complete HTML document (DOM) available for JavaScript to apply rendering logic on the pages accessed by users in this country.	ms	The value of this measure is the sum of the *Average DOM download time* and the *Average DOM processing time* measures. If the value of this measure is very high, then you may want to compare the *Average DOM download time* and the *Average DOM processing time* measures to figure out what is delaying DOM building – downloading? Or processing?

Measurement	Description	Measurement Unit	Interpretation
			A high value for this measure can adversely impact the *Average front end time*, which in turn can prolong the *Average page load time*. Ideally therefore, the value of this measure should be low.
Average DOM download time:	Indicates the time taken to download the complete HTML document for requests received from users in this country.	ms	Higher the download time of the document, longer will be the time taken to make the document available for page rendering. As a result, the overall user experience will be affected! This is why, a low value is desired for this measure at all times.
Average DOM processing time:	Indicates the time taken by the browser to build the Document Object Model (DOM) for the pages requested by the users in this country and make it available for JavaScript to apply rendering logic.	ms	An unusually high value for this measure is a clear indicator that DOM building is taking longer than normal. In consequence, page rendering will be delayed, thus adversely impacting user experience when users in this country are accessing the web site/web application. Ideally therefore, the value of this measure should be low.
Average first byte time:	Indicates the interval between the time that a user in this country initiates a request and the time that the browser receives the first response byte. In the context of an Ajax request, this is the interval between the Ajax request dispatch and the time that the browser receives the first response byte.	ms	Going by Figure 3.76 above, the *Average first byte time* is the time that elapsed between *navigationStart* and *responseStart*. The value of this measure is also the sum of *Average response available time*, *Average DNS lookup time*, and *Average TCP connection time*. This means that an abnormal increase in any of the above-mentioned time values will increase the value of this measure. If the first response byte from the target

Measurement	Description	Measurement Unit	Interpretation
			web site/web application is itself received slowly, it is bound to have a cascading effect on all events that follow - such as, document downloading, processing, and page rendering. Ultimately, this will impact the page load time as seen by users in this country. This is why, if the *Average first byte time* violates its threshold, administrators need to instantly switch to the troubleshooting mode and rapidly isolate what is causing it – is DNS lookup taking a long time? is the network connection to the web site/web application latent? or is the web server/web application server hosting the web site slow in processing requests? By comparing the values of the *Average response available time*, *Average DNS lookup time* , and *Average TCP connection time* measures, administrators can swiftly and accurately figure out the exact reason why there was a delay in receiving the first response byte. If this comparison reveals that the *Average DNS lookup time* is the highest, it implies that domain name resolution by the DNS server is taking a long time and impacting responsiveness. If the *Average TCP connection time* is found to be the culprit, then blame the network connection for delaying the transmission of the response byte. If the *Average reponse available time* is higher than the rest, you can be rest assured that the source of the problem lies with the server hosting the web site/web application.
Average response available time:	Indicates the interval between the start of	ms	In Figure 3.76 , the *Average response available* time is the time spent between

Measurement	Description	Measurement Unit	Interpretation
	processing of a request from a user in this country to when response is received.		the *requestStart* event and *responseStart* event. Ideally, a low value is desired for this measure, as high values will certainly hurt the *Apdex scrore* of the web site/web application. The key factor that can influence the value of this measure is the request processing ability of the web server/web application server that is hosting the web site/web application being monitored. Any slowdown in the backend web server/web application server - caused by the lack of adequate processing power in or improper configuration of the backend server - can significantly delay request processing by the server. In its aftermath, the *Average response available time* will increase, leaving users with an unsatisfactory experience with the web site/web application.
Average server connection time:	Indicates the elapsed time since a user in this country initiates a request to the web site/web application and the start of fetching the response document from it.	ms	In Figure 3.76, the time spent between *navigationStart* and *requestStart* makes up the *Average server connection time*. This includes the time to perform DNS lookups and the time to establish a TCP connection with the server. In other words, the value of this measure is nothing but the sum of the *Average DNS lookup time* and the *Average TCP connection time* measures. Ideally, the value of this measure should be low. A very high value will often end up delaying page loading and degrading the quality of the web site service. In the event that the server connection time is high therefore, simply compare the values

Measurement	Description	Measurement Unit	Interpretation
			of the *Average DNS lookup time* and *Average TCP connection time* measures to know to what this delay can be attributed - a delay in domain name resolution? Or a poor network connection to the server?
Average DNS lookup time:	Indicates the time taken by the browser used by a user in this country to perform the domain lookup for connecting to the web site/web application.	ms	A high value for this measure will not only affect DNS lookup, but will also impact the *Average server connection time* and *Average page load time*. This naturally will have a disastrous effect on user experience.
Average TCP connection time:	Indicates the time taken by the browser used by a user in this country to establish a TCP connection with the server.	ms	A bad network connection between the browser client and the server can delay TCP connections to the server As a result, the *Average server connection time* too will increase, thus impacting page load time and overall user experience with the web site/web application.

3.6.6.3 Cities Test

The **Cities** test provides an aggregate of the experience of users from different cities. If the users to your web site/web application are spread across different cities, you can use this test to know:

- Which cities your users are coming from;
- Which are the cities in which your web site/web application is very popular;
- How is the experience of users from each of these cities.

Moreover, if multiple users to a web site/web application are seeing slow page loads or error responses at around the same time, administrators can use the **Cities** test to figure out if the problem is specific to a particular city; if so, the test further pinpoints which exact city is impacted and why is the experience of users from that city poor – is the front end used by the users in this city inefficient? Is the WAN connection between the browser clients in this city and the backend server hosting the web site/web application latent? or is the problem with the backend? Detailed diagnostics provided by the test also point to the specific pages that are slow or have encountered JavaScript errors.

Target of the test : A web site/web application managed as a *Real User Monitor*

Agent deploying the test : A remote agent

Outputs of the test : One set of results for each browser used for accessing the web site/web application being monitored

Test parameters:

TEST PERIOD	How often should the test be executed
PROXYHOST	If the eG agent communicates with the RUM collector via a proxy, then specify the IP address/fully-qualified host name of the proxy server here. By default, this is set to *none*, indicating that the eG agent does not communicate with the collector via a proxy.
PROXYPORT	If the eG agent communicates with the RUM collector via a proxy, then specify the port at which the proxy server listens for requests from the eG agent. By default, this is set to *none*, indicating that the eG agent does not communicate with the collector via a proxy.
PROXYUSERNAME and **PROXYPASSWORD**	If the eG agent communicates with the RUM collector via a proxy server, and if proxy server requires authentication, then specify valid credentials for authentication against **PROXYUSERNAME** and **PROXYPASSWORD**. If no proxy server is used or if the proxy server used does not require authentication, then set the **PROXYUSERNAME** and **PROXYPASSWORD** to *none*. **Note:** If you **Reconfigure** the test later to change the values of the **PROXYUSERNAME**, **PROXYPASSWORD**, and **CONFIRM PASSWORD** parameters, then such changes will be effected only if the eG remote agent monitoring the Real User Monitor component is restarted.
CONFIRM PASSWORD	Confirm the **PROXYPASSWORD** by retyping it here.
DO YOU WANT TO LIMIT THE PAGE VIEWS?	By default, the eG RUM monitors all requests to a managed web site. This is why, this flag is set to **No** by default. However, in case of web sites that receive thousands of hits every day, monitoring each page view may add significantly to the overhead of the eG agent and may also increase the size of the eG database considerably. To reduce the strain on both the eG agent and the eG backend, you may want to restrict the monitoring scope of this test to a few page visits. To achieve this, first set this flag to **Yes**. This will invoke the option depicted by the below figure.  Figure 3.84: Configuring the number of allowed page visits By default, the **MAXIMUM ALLOWED PAGE VISITS PER DAY** is set to *100000*. This implies that the test will consider only the first 100000 requests in a day for monitoring. All page visits beyond 100000 will by default be excluded from the test's monitoring purview. You can increase or decrease this limit, if you so need.
URL SEGMENTS TO BE USED AS GROUPED URL	**This parameter is applicable to the Page Groups test alone**. The **Page Groups** test groups URLs based on the URL segments configured for monitoring and reports aggregated response time metrics for every group. Using this parameter, you can specify a comma-separated list of URL segment

	numbers based on which the pages are to be grouped. URL segments are the parts of a URL (after the base URL) or path delimited by slashes. So if you had the URL: http://www.eazykart.com/web/shopping/login.jsp , then http://www.eazykart.com will be the base URL or domain, */web* will be the first URL segment, */shopping* will be the second URL segment, and */login.jsp* will be the third URL segment. By default, this parameter is set to *1,2.* This default setting, when applied to the sample URL provided above, implies that the eG agent will aggregate request and response time metrics to all instrumented web pages (i.e., web pages with the code snippet) under the URL */web/shopping*. Here, */web* corresponds to the specification 1 (URL segment 1) and */shopping* corresponds to the specification 2 (URL segment 2) in the default value *1,2*. This in turn means that, if the web site consists of pages such as http://www.eazykart.com/web/shopping/products.jsp , http://www.eazykart.com/web/shopping/products/travel/bags.jsp , http://www.eazykart.com/web/shopping/payment.jsp, etc., then the eG agent will track the requests to and responses from all these web pages, aggregate the results, and present the aggregated metrics for the descriptor /web/shopping. This way, the test will create different page groups based on each of the second-level URL segments in the managed web site – eg., */web/movies* , */web/travel* , */web/reservations* , */partner/contacts* etc. – and will report aggregated metrics for each group so created. If you want, you can override the default setting by providing different URL segment numbers here. For instance, your specification can be just *2*. In this case, for the URL http://www.eazykart.com/web/shopping/login.jsp, the test will report metrics for the descriptor */shopping. You can even set this parameter to 1,3. For a web site that contains URLs such as http://www.eazykart.com/web/shopping/products.jsp* , http://www.eazykart.com/web/shopping/products/travel/bags.jsp , and http://www.eazykart.com/web/shopping/payment.jsp*,* this specification will result in the following descriptors: */web/products*, */web/products.jsp, and /web/payment.jsp.*
URL PATTERNS TO BE IGNORED FROM MONITORING	By default, this test does not track requests to the following URL patterns: **.js,*.css,*.jpeg,*.jpg,*.png*. If required, you can remove one/more patterns from this default list, so that such patterns are monitored, or can append more patterns to this list in order to exclude them from monitoring. For instance, to additionally ignore URLs that end with *.gif* and *.bmp* when monitoring, you need to alter the default specification as follows: **.js,*.css,*.jpeg,*.jpg,*.png,*.gif,*.bmp* **Note:** The URL patterns configured here are not just used by the eG agent to filter out unimportant performance data during metrics collection; these patterns are also used by the RUM collector to determine which beacons should be accepted and which ones need to be discarded. Typically, the very first time this test runs and polls the RUM collector for metrics, the eG agent executing this test searches the performance records stored in the collector

	for data that pertains to the URL patterns configured for exclusion. If such data is found, the agent then ignores that data during metrics collection. This means that during the very first test execution, URL filtering is performed only by the eG agent. During this time, the RUM collector downloads the URL patterns configured against this parameter from the eG agent. Armed with this information, the collector then scans all beacons that browsers send subsequently, and determines if there are any beacons for the excluded URL patterns. If such beacons are found, the collector discards them instantly. Filtering URLs at the collector-level significantly reduces the load on the collector and conserves storage space on the collector; it also minimizes the workload of the eG agent. By additionally filtering URLs at the agent-level, eG makes sure that even if beacons pertaining to excluded URL patterns find their way into the RUM collector, they are captured and siphoned out by the eG agent.
JAVASCRIPT ERRORS TO BE IGNORED	By default, this test alerts administrators to all Javascript errors that occur in the monitored web site/web application. This is why, this parameter is set to *none* by default. Sometimes however, administrators may not want to be notified when certain types of Javascript errors occur – this could be because such errors are harmless or are a normal occurrence in their environment. In such circumstances, you can instruct the eG agent to ignore these errors when monitoring. For this, specify the Java script error message to be ignored in the **JAVASCRIPT ERRORS TO BE IGNORED** text box, in the following format: *<Javascript eror message>:<URL of the page/file where the message originated>*. For instance, say that the *login.html* page in your web site runs a few Java scripts that throw *Object expected* errors, which you want the eG agent to ignore. In this case, your error specification can be as follows: *Object expected:http://www.eazykart.com/web/login.html*. Alternatively, you can provide only that text string with which the error message begins in your specification – eg., *Object:http://www.eazykart.com/web/login.html*. Moreover, instead of the complete URL, you can specify just the name of the HTML/jsp/aspx page in which the error is to be ignored – example: *Object:login.html*. Sometimes, the individual web pages in your web site may not run any Java script directly. Instead, these web pages may include links to Java script files that will run the Java script and return the output to the web pages. If you want the eG agent to ignore certain errors thrown by such a Javascript file, then your error pattern specification should include the URL of the Javascript file and not the web page that points to it. This is because, in this case, the file is where the error message originates. For instance, in the same example above, if the *login.html* page points to a *validate.js* file, and you want to ignore the *Object expected errors* that this JS file throws, your error pattern specification will either be, *Object expected:validate.js*, or *Object:validate.js*. Multiple error message-URL combinations can also be provided as a comma-separated list. The format of your specification will be: *<Javascript error message 1>:<Originating URL 1>,<Javascript error message 2>:<Originating URL 2>,. . .*

	For example, to ignore the *Object expected* and *Uncaught TypeError* errors in the *login.html* page, use the following specification: *Object:login.html,Uncaught:login.html* Likewise, to ignore the *Object expected* error in the *login.html* page and the *Uncaught TypeError* in the *validate.js* file, your specification will be: *Object:login.html*,*Uncaught:validate.js* If you want to ignore the *Uncaught TypeError* across all the pages of the web site, your specification will be as follows: *Uncaught TypeError:All* **Note:** When specifying the <*Javascript error message*> to be ignored, take care of the following: • The error message should not contain any special characters – in particular, the ':' (the colon) and the ',' (the comma) characters should be avoided. • The case of the actual error message and the one specified as part of your error specification should match. This is because, the eG agent performs *case-sensitive* pattern matching.
MAXIMUM HEALTHY TRANSACTIONS IN DD	By default, this parameter is set to *5* indicating that the detailed diagnosis of this test will collect and display metrics related to the top-5 normal/healthy page views, in terms of the *Average page load time*. To identify the top 5, the eG agent sorts all healthy transactions in the ascending order of their *Average page load time*, and picks the first 5 transactions from this sorted list for display in the **DETAILED DIAGNOSIS** page and for storing in the eG database. You can however, increase or decrease this value depending upon how many healthy transactions you want to see in your detailed diagnosis, and how well-tuned or well-sized the eG database is. If you do not want the detailed diagnosis to include any healthy transaction, set the value of this parameter to *0*. To view all healthy transactions, set the value of this parameter to *all*. Before setting the value of this parameter to *all*, make sure that you have a well-sized eG database in place, as this setting will store details of every transaction that registers an *Avg page load time* value that is lower than the **SLOW TRANSACTION CUTOFF** specification. On a good day, this can result in numerous transactions, and can consume considerable space in the eG database.
MAXIMUM SLOW TRANSACTIONS IN DD	By default, this parameter is set to *5* indicating that the detailed diagnosis of this test will display metrics related to all the top-5 slow transactions, in terms of the *Average page load time*. To identify the top 5, the eG agent sorts all slow transactions in the descending order of their *Average page load time*, and picks the first 5 transactions from this sorted list for display in the **DETAILED DIAGNOSIS** page and for storing in the eG database.

	You can however, increase or decrease this value depending upon how many slow transactions you want to see in your detailed diagnosis, and how well-tuned or well-sized the eG database is. If you do not want the detailed diagnosis to include any slow transaction, set the value of this parameter to *0*. To view all slow transactions, set the value of this parameter to *all*. Before setting the value of this parameter to *all*, make sure that you have a well-sized eG database in place, as this setting will store details of every transaction that registers an *Avg page load time* value that is higher than the **SLOW TRANSACTION CUTOFF** specification. On a bad day, this can result in numerous transactions, and can consume considerable space in the eG database.
MAXIMUM ERROR TRANSACTIONS IN DD	By default, this parameter is set to *5*, indicating that the detailed diagnosis of this test will display metrics related to the top-5 transactions that encountered JavaScript errors, based on when those errors occurred. To identify the top 5, the eG agent sorts all error transactions in the descending order of the date/time at which the errors occurred, and picks the first 5 transactions from this sorted list for display in the **DETAILED DIAGNOSIS** page and for storing in the eG database. You can however, increase or decrease this value depending upon how many error transactions you want to see in your detailed diagnosis, and how well-tuned or well-sized the eG database is. If you do not want the detailed diagnosis to include any error transaction, set the value of this parameter to *0*. To view all error transactions, set the value of this parameter to *all*. Before setting the value of this parameter to *all*, make sure that you have a well-sized eG database in place, as this setting will store details of every transaction that encounters a JavaScript error. On a bad day, this can result in numerous transactions, and can consume considerable space in the eG database.
SLOW TRANSACTION CUTOFF (MS)	This test reports the count of slow page views and also pinpoints the pages that are slow. To determine whether/not a page is slow, this test uses the **SLOW TRANSACTION CUTOFF** parameter. By default, this parameter is set to *4000 millisecs* (i.e., 4 seconds). This means that, if a page takes more than 4 seconds to load, this test will consider that page as a slow page by default. You can increase or decrease this slow transaction cutoff according to what is 'slow' and what is 'normal' in your environment. **Note:** The default value of this parameter is the same as the default *Maximum threshold* setting of the *Avg page load time* measure – i.e., both are set to *4000 millisecs* by default. While the former helps eG to distinguish between slow and healthy page views for the purpose of providing detailed diagnosis, the latter tells eG when to generate an alarm on *Avg page load time*. For best results, it is recommended that both these settings are configured with the same value at all times. Therefore, if you change the value of one of these configurations, then make sure you update the value of the other as well. For instance, if the **SLOW TRASACTION CUTOFF** is changed to *6000 millisecs*, change the *Maximum Threshold* of the *Avg page load time* measure to *6000* millsecs as well.

SEND ZERO VALUES WHEN THERE IS NO TRAFFIC	By default, this flag is set to **No**. This implies that, if there is no traffic to a monitored web site/web application – i.e., if all measures of this test return only the value *0* - then the eG agent will not report these metrics to the eG manager. This also means that, by default, users to the eG monitoring console will not know that there is no traffic to the web site/web application. You can however, ensure that users to the eG monitoring console are informed of the absence of any user activity on the web site/web application. For this, set this flag to **Yes**. If this is done, then the eG agent will report all the metrics of this test to the eG manager, despite the fact that their value is *0*. These zero values will clearly indicate to users that there no traffic to the monitored web site/web application.
PAGELOADTIME CUTOFF (MS)	eG RUM relies on the Navigation API to track page views and capture page load time. Sometimes, the Navigation API can incorrectly report abnormally high values for page load time. To ensure that the eG agent ignores beacons with such page load time values, you can configure in this text box, the maximum value for page load time. By default, this is set to 3600000 (ms). This implies that by default, the eG agent will disregard all beacons that carry page load time values higher than 3600000 milliseconds. You can change this value to suit your environment.
PAGE TYPES TO BE INCLUDED IN DASHBOARD	By default, the eG RUM Dashboard displays details of base page views only. You can optionally include Ajax and/or iFrame views as well in the dashboard, by selecting the relevant options from this list box.
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**.
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.

Measurements made by the test

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Page views:	Indicates the total number of times pages were viewed by users from this city.	Number	This is a good measure of the traffic from a specific city. Sudden, but significant spikes in the page view count could be a cause for concern, as it could be owing to a malicious virus attack or an unscrupulous attempt to hack your web site/web application. **Note:** An abnormally high value for this measure may not always be a cause for concern; nor would it always indicate a genuine increase in traffic to the web site/web application. If the eG agent monitoring the *Real User Monitor* component is stopped for sometime (say, for maintenance purposes) and then started, or if the eG agent-collector connection breaks and is restored after a while, the eG agent will pull all the metrics that the collector stored locally during the period of its absence, cumulate them, and then display the cumulated values in the eG monitoring console as metrics that pertain to the current measurement period. In reality, these metrics pertain to the entire time period when the eG agent was unavailable. Because of this, the Page views measure may indicate a sudden and significant surge in traffic.
Apdex score:	Indicates the apdex score of the web site/web application based on the experience of users from this city.	Number	**Apdex** (Application Performance Index) is an open standard developed by an alliance of companies. It defines a standard method for reporting and comparing the performance of software applications in computing. Its purpose is

Measurement	Description	Measurement Unit	Interpretation
			to convert measurements into insights about user satisfaction, by specifying a uniform way to analyze and report on the degree to which measured performance meets user expectations. The Apdex method converts many measurements into one number on a uniform scale of 0-to-1 (0 = no users satisfied, 1 = all users satisfied). The resulting Apdex score is a numerical measure of user satisfaction with the performance of enterprise applications. This metric can be used to report on any source of end- user performance measurements for which a performance objective has been defined. The Apdex formula is: *Apdext = (Satisfied Count + Tolerating Count / 2) / Total Samples* This is nothing but the number of satisfied samples plus half of the tolerating samples plus none of the frustrated samples, divided by all the samples. A score of 1.0 means all responses were satisfactory. A score of 0.0 means none of the responses were satisfactory. Tolerating responses half satisfy a user. For example, if all responses are tolerating, then the Apdex score would be 0.50. Ideally therefore, the value of this measure should be 1.0. A value less than 1.0 indicates that the experience of users from this city has been less than satisfactory.
Average page load	Indicates the average time	ms	This is the average interval between the

Measurement	Description	Measurement Unit	Interpretation
time:	that pages accessed by users from this city took to load completely on the browser.		time that a user initiates a request and the completion of the page load of the response in the user's browser. In the context of an Ajax request, it ends when the response has been completely processed. By comparing the value of this measure across cities, you will be able to tell if the page load time is significantly higher for any one city – this is the first sign that the problem could be city-specific. Next, check the detailed diagnosis of this measure for this city to find out which pages are slow. Then, look up the detailed measures for the other cities to ascertain whether users from those cities also experienced slowness when accessing the same pages. If not, it is a sure sign that the problem is specific to the city. So, proceed to compare the values of the *Average front end time, Average server connection time*, and *Average response available time* measures for that city, to know why the users in that city are seeing slowness – is it the front end? network? or the backend? If the *Average front end time is the maximum*, then the problem is the front end used by the users in that city. If the *Average server connection time* is the highest, then the network connection between the user's browser and the web site/web application is the one contributing to the slowness. A very high *Average response available time* on the other hand, reveals that problem may have nothing to do with the city, but with the server that is hosting the web

Measurement	Description	Measurement Unit	Interpretation
			site/web application.
Unique user session:	Indicates the number of distinct users who are currently accessing the web site/web application from this city.	Number	
Page views per minute:	Indicates the number of times the pages were viewed per minute by users from this city.	Number	An unusually high value for this measure may require investigation.
Normal page view percentage:	Indicates the percentage of page views that delivered a satisfactory experience to users from this city.	Percent	The value of this measure indicates the percentage of page views in which users from this city have neither experienced any slowness, nor encountered any Javascript errors. Ideally, the value of this measure should be 100%. A value that is slightly less than 100% indicates that the user experience has not been up to the mark. A value less than 50% is indicative of a serious problem, where most of the page views are either slow or have encountered Javascript errors. Under such circumstances, to know what exactly is affecting the experience of users, compare the value of the *Slow page view percentage* with that of the Javasript error page view percentage for that browser. This will reveal the reason for the poor user experience – slow pages? or Javascript errors? If slow pages are the problem, use the detailed diagnosis of the Slow page views measure to know which pages are slow and where these pages are losing time – in the front end? network? or backend?.

Measurement	Description	Measurement Unit	Interpretation
			If JavaScript errors are the problem, use the detailed diagnosis of the JavaScript error view percentage measure to know what errors occurred in which pages.
Slow page view percentage:	Indicates the percentage of pages that were slow in loading when accessed by users in this city.	Percent	Ideally, the value of this measure should be 0. A value over 50% implies that you are in a spot of bother, with over half of the page views being slow. Use the detailed diagnosis of the Slow page views measure to identify the slow pages and isolate the root-cause of the slowness – is it the front end? the network? or the backend?
JavaScript error view percentage:	Indicates the percentage of page views that have encountered JavaScript errors.	Percent	Ideally, the value of this measure should be 0. A value over 50% implies that you are in a spot of bother, with over half of the page views experiencing JavaScript errors.
Slow page views (Tolerating & Frustrated):	Indicates the number of page views that were slow when accessed from this city.	Number	A page view is considered to be slow when the average time taken to load that page exceeds the slow transaction cutoff configured for this test. Ideally, a page should load quickly. The value 0 is hence desired for this measure. If the value of this measure is high, it indicates that users from this city frequently experienced slowness when accessing pages in the web site/web application. To know which page views are slow and why, use the detailed diagnosis of this measure.
JavaScript error page views:	Indicates the number of times JavaScript errors occurred when accessing pages from this city.	Number	Ideally, the value of this measure should be 0. A high value indicates that many JavaScript errors are occurring when accessing pages in the web site/web

Measurement	Description	Measurement Unit	Interpretation
			application. Use the detailed diagnosis of this measure to identify the error pages and to know what Javascript error has occurred in which page. This will greatly aid troubleshooting!
Satisfied page views:	Indicates the number of times pages were viewed by users in this city without any slowness.	Number	A page view is considered to be slow when the average time taken to load that page exceeds the slow transaction cutoff configured for this test. If this slow transaction cutoff is not exceeded, then the page view is deemed to be 'satisfactory'. To know which page views are satisfactory, use the detailed diagnosis of this measure. Ideally, the value of this measure should be the same as that of the Page views measure. If not, then it indicates that one/more page views are slow – i.e., have violated the slow transaction cutoff. If the value of this measure is much lesser than the value of the *Tolerating page views* and the *Frustrated page views*, it is a clear indicator that the experience of users in this city has been below-par. In such a case, use the detailed diagnosis of the *Tolerating page views* and *Frustrated page views* measures to know which pages are slow and why.
Tolerating page views:	Indicates the number of tolerating page views experienced by users in this city.	Number	If the *Average page load time* of a page exceeds the slow transaction cutoff configuration of this test, but is less than 4 times the slow transaction cutoff (i.e., < 4 * slow transaction cutoff), then such a page view is considered to be a Tolerating page view.

Measurement	Description	Measurement Unit	Interpretation
			Ideally, the value of this measure should be 0. A value higher than that of the *Satisfied page views* measure is a cause for concern, as it implies that the overall experience of the users in this city is less than satisfactory. To know which pages are contributing to this sub- par experience, use the detailed diagnosis of this measure. The detailed metrics will also enable you to accurately isolate what is causing the tolerating page views – a problem with the front end? network? or backend?
Frustrated page views:	Indicates the number of frustrated page views experienced by users in this city.	Number	If the Average page load time of a page is over 4 times the slow transaction cuttoff configuration of this test (i.e., > 4 * slow transaction cutoff), then such a page view is considered to be a Frustrated page view. Ideally, the value of this measure should be 0. A value higher than that of the *Satisfied page views* measure is a cause for concern, as it implies that the experience of users in this city has been less than satisfactory. To know which pages are contributing to this sub- par experience, use the detailed diagnosis of this measure. The detailed metrics will also enable you to accurately isolate what is causing the frustrated page views – a problem with the front end? network? or backend?
Average front end time:	Indicates the interval between the arrival of the first byte of text response and the completion of the response page rendering by the browser used by	ms	In the Figure 3.76 that depicts a typical page loading process, the Average front end time denotes the time from the *responseStart* event to the *loadEventEnd* . This process includes document downloading, processing, and

Measurement	Description	Measurement Unit	Interpretation
	users in this city.		page rendering. This time is therefore the sum of the Average DOM ready time and the *Average page rendering time*. If the *Average page load time* exceeds its threshold, then you may want to compare the value of this measure with that of the *Average server connection time* and *Average response available time* to zoom into the source of the slowness - is it the front end? the network? or the backend?
Average page rendering time:	Indicates the time taken to complete the download of remaining resources, including images, and to finish rendering the pages in the browser for users in this city.	ms	A high value of this measure indicates that the pages are taking too long to be rendered. This can adversely impact the *Average front end time*, which in turn can prolong the *Average page load time*. Ideally therefore, the value of this measure should be low.
Average DOM ready time:	Indicates the time taken by the browser to make the complete HTML document (DOM) available for JavaScript to apply rendering logic on the pages accessed by users in this city.	ms	The value of this measure is the sum of the Average DOM download time and the *Average DOM processing time* measures. If the value of this measure is very high, then you may want to compare the *Average DOM download time* and the *Average DOM processing time* measures to figure out what is delaying DOM building – downloading? Or processing? A high value for this measure can adversely impact the *Average front end time*, which in turn can prolong the *Average page load time*. Ideally therefore, the value of this measure should be low.
Average DOM download time:	Indicates the time taken to download the complete HTML document for requests received from users in this city.	ms	Higher the download time of the document, longer will be the time taken to make the document available for page rendering. As a result, the overall user experience will be affected! This is why, a

Measurement	Description	Measurement Unit	Interpretation
			low value is desired for this measure at all times.
Average DOM processing time:	Indicates the time taken by the browser to build the Document Object Model (DOM) for the pages requested by the users in this city and make it available for JavaScript to apply rendering logic.	ms	An unusually high value for this measure is a clear indicator that DOM building is taking longer than normal. In consequence, page rendering will be delayed, thus adversely impacting user experience when users in this city are accessing the web site/web application. Ideally therefore, the value of this measure should be low.
Average first byte time:	Indicates the interval between the time that a user in this city initiates a request and the time that the browser receives the first response byte. In the context of an Ajax request, this is the interval between the Ajax request dispatch and the time that the browser receives the first response byte.	ms	Going by Figure 3.76 above, the *Average first byte time* is the time that elapsed between *navigationStart* and *responseStart*. The value of this measure is also the sum of *Average response available time* , *Average DNS lookup time*, and *Average TCP connection time*. This means that an abnormal increase in any of the above-mentioned time values will increase the value of this measure. If the first response byte from the target web site/web application is itself received slowly, it is bound to have a cascading effect on all events that follow – such as, document downloading, processing, and page rendering. Ultimately, this will impact the page load time as seen by users in this city. This is why, if the *Average first byte time* violates its threshold, administrators need to instantly switch to the troubleshooting mode and rapidly isolate what is causing it – is DNS lookup taking a long time? is the network connection to the web site/web application latent? or is the web server/web application server hosting the

Measurement	Description	Measurement Unit	Interpretation
			web site slow in processing requests? By comparing the values of the *Average response available time, Average DNS lookup time* , and *Average TCP connection time* measures, administrators can swiftly and accurately figure out the exact reason why there was a delay in receiving the first response byte. If this comparison reveals that the *Average DNS lookup time* is the highest, it implies that domain name resolution by the DNS server is taking a long time and impacting responsiveness. If the *Average TCP connection time* is found to be the culprit, then blame the network connection for delaying the transmission of the response byte. If the *Average reponse available time* is higher than the rest, you can be rest assured that the source of the problem lies with the server hosting the web site/web application.
Average response available time:	Indicates the interval between the start of processing of a request from a user in this city to when response is received.	ms	In Figure 3.76 , the *Average response available time* is the time spent between the *requestStart* event and *responseStart* event. Ideally, a low value is desired for this measure, as high values will certainly hurt the Apdex scrore of the web site/web application. The key factor that can influence the value of this measure is the request processing ability of the web server/web application server that is hosting the web site/web application being monitored. Any slowdown in the backend web server/web application server – caused

Measurement	Description	Measurement Unit	Interpretation
			by the lack of adequate processing power in or improper configuration of the backend server – can significantly delay request processing by the server. In its aftermath, the *Average response available time* will increase, leaving users with an unsatisfactory experience with the web site/web application.
Average server connection time:	Indicates the elapsed time since a user in this city initiates a request to the web site/web application and the start of fetching the response document from it.	ms	In Figure 3.76, the time spent between *navigationStart* and *requestStart* makes up the *Average server connection time*. This includes the time to perform DNS lookups and the time to establish a TCP connection with the server. In other words, the value of this measure is nothing but the sum of the *Average DNS lookup time* and the *Average TCP connection time* measures. Ideally, the value of this measure should be low. A very high value will often end up delaying page loading and degrading the quality of the web site service. In the event that the server connection time is high therefore, simply compare the values of the *Average DNS lookup time* and *Average TCP connection time* measures to know to what this delay can be attributed – a delay in domain name resolution? Or a poor network connection to the server?
Average DNS lookup time:	Indicates the time taken by the browser used by a user in this city to perform the domain lookup for connecting to the web site/web application.	ms	A high value for this measure will not only affect DNS lookup, but will also impact the *Average server connection time* and *Average page load time*. This naturally will have a disastrous effect on user experience.

Measurement	Description	Measurement Unit	Interpretation
Average TCP connection time:	Indicates the time taken by the browser used by a user in this city to establish a TCP connection with the server.	ms	A bad network connection between the browser client and the server can delay TCP connections to the server. As a result, the *Average server connection time* too will increase, thus impacting page load time and overall user experience with the web site/web application.

Note:

- For best results, all the tests mapped to the *Real User Monitor* component have been programmed to run at the same frequency. This is why, if you change the **TEST PERIOD** of one test, eG automatically updates the **TEST PERIOD** of all the other tests as well. However, if you exclude a particular test for a *Real User Monitor* component, change the **TEST PERIOD** of one of the other tests, and then include the test that was earlier excluded, the test that was just included will not be updated with the **TEST PERIOD** change that was made before. As a result, you will have a single RUM test alone running at a frequency that is different from the rest. This test may hence report inaccurate metrics.
- If you increase the **TEST PERIOD** of any test, then remember to increase the value of the **MAXIMUM ALLOWED PAGE VISITS PER DAY** parameter as well. This is because, increasing the test frequency means that more data will now be fetched from the RUM collector by the eG agent. If the **MAXIMUM ALLOWED PAGE VISITS PER DAY** still remains low at this point, then the eG agent will ignore many of the performance beacons stored in the RUM collector at the time of metrics collection.
- By default, the **Web Site** test will report the value 0 for all metrics, if there is no traffic to the web site/web application being monitored. All other RUM tests on the other hand will not report any measures at all, in the event of zero traffic. Now, say you have set a *Minimum threshold* of 1 for the *Page views* measure of the **Cities** test. This typically means that you want to be alerted if no pages are viewed - i.e., if there is no traffic to the web site. However, because the **Cities** test does not report any metrics at all during no-traffic conditions, the eG Enterprise system will not send out any alert if the *Minimum threshold* of this test is violated. To enable the alerting capability for the **Cities** test during the periods when there is no traffic, you will first have to enable the test to report the value 0 for metrics when the web site does not receive any page view requests. For this, you will first have to set the **SEND ZERO VALUES WHEN THERE IS NO TRAFFIC** flag of the **Cities** test to **Yes**.

3.6.7 Detailed Diagnostics

The real-time measures reported by the eG Real User Monitor provide a high-level view of user experience. In other words, the measures offer a broad web site-level, page group-level, or device-level perspective to web

site/web application performance. To provide more granular, page-level insight into the user experience with a web site/web application, the eG Real User Monitor collects detailed diagnostics at configured intervals.

By default, it is the **Web site** test that collects these additional heuristics. The metrics so collected are available as part of the detailed diagnosis of the following measures:

- Page views
- Average page load time
- JavaScript error page views
- JavaScript error page view percentage
- Slow page views
- Frustrated page views
- Tolerating page views
- Desktop page views
- Mobile page views
- Tablet page views

The basic information provided in the detailed diagnosis remains the same for all the measures listed above. Figure 3.85 below provides a sneak-peek at the standard columns that are available as part of the detailed diagnosis of all measures.

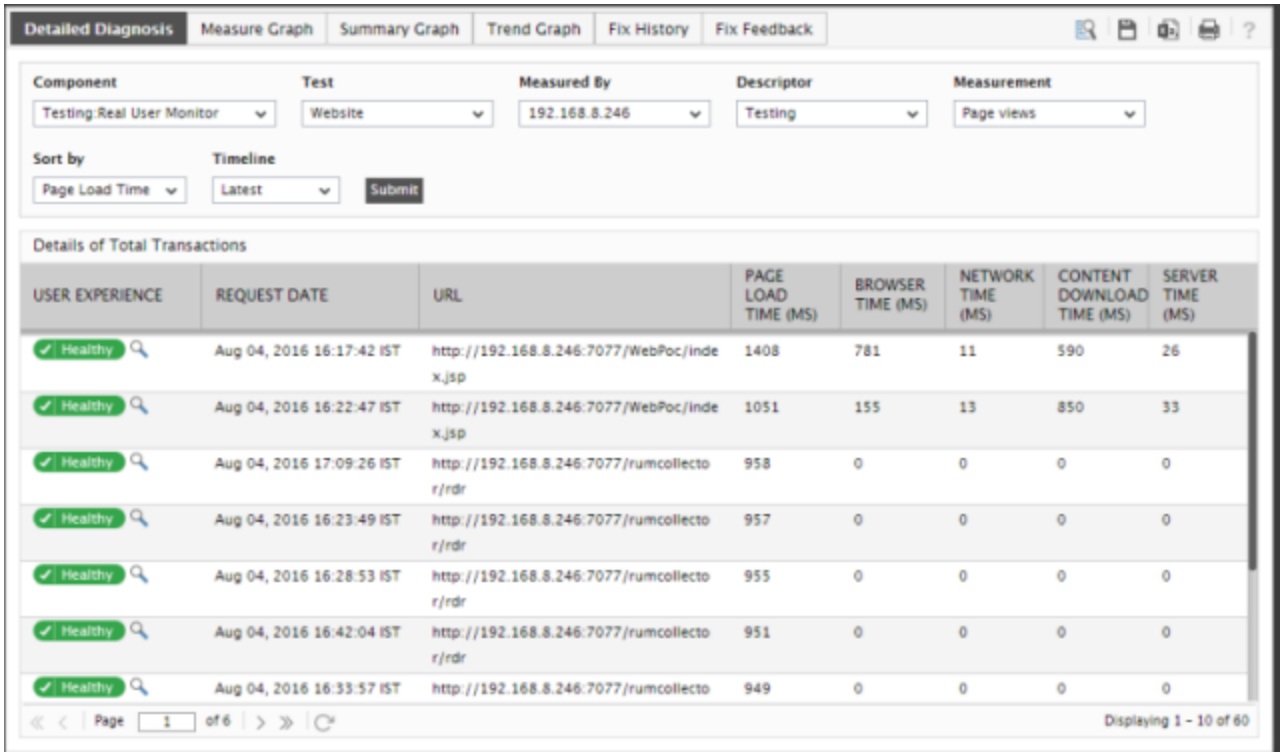


Figure 3.85: Sample detailed diagnosis of a measure reported by the Web site test

This basic information includes the following:

- Which pages were accessed and at what times; this will point you to those pages that were accessed frequently;
- What is the user experience with each page view (slow? healthy? Or error?), the average load time per page view, and the break-up of the load time of every page view; slow page views can thus be quickly identified and the reason for the slowness accurately isolated - is it the browser? the network? content download? or the server?

What pages are listed in the detailed diagnosis will however, differ according to the following:

- The measure with which the detailed diagnosis is associated;
- The configuration of the following parameters of the **Web site** test:
  - **MAXIMUM HEALTHY TRANSACTIONS IN DD**
  - **MAXIMUM SLOW TRANSACTIONS IN DD**
  - **MAXIMUM ERROR TRANSACTIONS IN DD**

  Using these parameters, you can dictate how much information is to be collected, stored, and reported as part of detailed diagnosis. By default, all these parameters are set to *5*. You can increase or decrease the value of these parameters to display more or less number of page views in your detailed diagnosis.

This means that, by default, the detailed diagnosis of the **Web site** test will display the following for each measure reported for every URL Segment :

Page view

- The top-5 page views accessed recently, which registered a satisfactory user experience;
- The top-5 slow page views, in terms of the page load time;
- The top-5 page views that recently encountered JavaScript errors;

Details of Total Transactions

USER EXPERIENCE	REQUEST DATE	URL	PAGE LOAD TIME (MS)	BROWSER TIME (MS)	NETWORK TIME (MS)	CONTENT DOWNLOAD TIME (MS)	SERVER TIME (MS)
Slow	Aug 18, 2016 01:44:50 EDT	http://192.168.11.121:8780/EasyKart/PaymentPage.jsp	30126	1816	201	2462	25647
Slow	Aug 18, 2016 02:29:56 EDT	http://192.168.11.121:8780/EasyKart/PaymentPage.jsp	30126	1816	201	2462	25647
Slow	Aug 18, 2016 01:44:50 EDT	http://192.168.11.121:8780/EasyKart/PaymentPage.jsp	30017	1808	201	2454	25554
Slow	Aug 18, 2016 02:29:56 EDT	http://192.168.11.121:8780/EasyKart/PaymentPage.jsp	30017	1808	201	2454	25554
Slow	Aug 18, 2016 01:44:50 EDT	http://192.168.11.121:8780/EasyKart/PaymentPage.jsp	29890	1801	199	2443	25447
Slow	Aug 18, 2016 02:29:56 EDT	http://192.168.11.121:8780/EasyKart/PaymentPage.jsp	29890	1801	199	2443	25447
Slow	Aug 18, 2016 01:54:50	http://192.168.11.121:8780/EasyKart/Ship	29851	1798	199	2441	25413

Page 1 of 7 Displaying 1 - 10 of 68

Figure 3.86: The detailed diagnosis of the Page views measure

Average page load time

- The top-5 slow page views, in terms of the page load time;
- The top-5 page views that recently encountered JavaScript errors;

Details of Slow Transactions

USER EXPERIENCE	REQUEST DATE	URL	PAGE LOAD TIME (MS)	BROWSER TIME (MS)	NETWORK TIME (MS)	CONTENT DOWNLOAD TIME (MS)	SERVER TIME (MS)
Slow	Aug 18, 2016 01:44:50 EDT	http://192.168.11.121:8780/EasyKart/PaymentPage.jsp	30126	1816	201	2462	25647
Slow	Aug 18, 2016 01:44:50 EDT	http://192.168.11.121:8780/EasyKart/PaymentPage.jsp	30017	1808	201	2454	25554
Slow	Aug 18, 2016 01:44:50 EDT	http://192.168.11.121:8780/EasyKart/PaymentPage.jsp	29890	1801	199	2443	25447
Slow	Aug 18, 2016 01:54:50 EDT	http://192.168.11.121:8780/EasyKart/ShippingPage.jsp	29851	1798	199	2441	25413
Slow	Aug 18, 2016 01:54:50 EDT	http://192.168.11.121:8780/EasyKart/AddToCart.jsp	29571	1781	199	2418	25173
Slow	Aug 18, 2016 01:44:50 EDT	http://192.168.11.121:8780/EasyKart/BrowseProducts.jsp	29548	1780	199	2416	25153
Slow	Aug 18, 2016 02:04:50	http://192.168.11.121:8780/EasyKart/AddT	29528	1779	199	2414	25136

Page 1 of 6 Displaying 1 - 10 of 55

Figure 3.87: The detailed diagnosis of the Avg page load time measure

JavaScript error view percentage and JavaScript error page views

The top-5 page views that recently encountered JavaScript errors;

Details of Error Transactions

USER EXPERIENCE	REQUEST DATE	URL	PAGE LOAD TIME (MS)	BROWSER TIME (MS)	NETWORK TIME (MS)	CONTENT DOWNLOAD TIME (MS)	SERVER TIME (MS)
Error	Aug 18, 2016 02:59:54 EDT	http://192.168.11.121:8780/corebanking/retail/loginaction.jsp	80146	4829	255	6552	68510
Error	**Aug 18, 2016 02:24:54 EDT**	**http://192.168.11.121:8780/corebanking/retail/loginaction.jsp**	**80146**	**4829**	**255**	**6552**	**68510**
Error	Aug 18, 2016 02:59:54 EDT	http://192.168.11.121:8780/corebanking/retail/beneficiarydetails.jsp	60325	3635	233	4932	51525
Error	Aug 18, 2016 02:24:54 EDT	http://192.168.11.121:8780/corebanking/retail/beneficiarydetails.jsp	60325	3635	233	4932	51525
Error	Aug 18, 2016 02:14:57 EDT	http://192.168.11.121:8780/corebanking/retail/doorstepbankingservice.jsp	34567	2083	205	2826	29453
Error	Aug 18, 2016 02:49:54 EDT	http://192.168.11.121:8780/corebanking/retail/doorstepbankingservice.jsp	34567	2083	205	2826	29453

Figure 3.88: The detailed diagnosis of the JavaScript errors measure

Slow page views

- The top-5 slow page views, in terms of the page load time;

Details of Slow Transactions

USER EXPERIENCE	REQUEST DATE	URL	PAGE LOAD TIME (MS)	BROWSER TIME (MS)	NETWORK TIME (MS)	CONTENT DOWNLOAD TIME (MS)	SERVER TIME (MS)
Slow	**Aug 18, 2016 01:44:50 EDT**	**http://192.168.11.121:8780/EasyKart/PaymentPage.jsp**	**30126**	**1816**	**201**	**2462**	**25647**
Slow	Aug 18, 2016 02:29:56 EDT	http://192.168.11.121:8780/EasyKart/PaymentPage.jsp	30126	1816	201	2462	25647
Slow	Aug 18, 2016 01:44:50 EDT	http://192.168.11.121:8780/EasyKart/PaymentPage.jsp	30017	1808	201	2454	25554
Slow	Aug 18, 2016 02:29:56 EDT	http://192.168.11.121:8780/EasyKart/PaymentPage.jsp	30017	1808	201	2454	25554
Slow	Aug 18, 2016 01:44:50 EDT	http://192.168.11.121:8780/EasyKart/PaymentPage.jsp	29890	1801	199	2443	25447
Slow	Aug 18, 2016 02:29:56 EDT	http://192.168.11.121:8780/EasyKart/PaymentPage.jsp	29890	1801	199	2443	25447
Slow	Aug 18, 2016 01:54:50	http://192.168.11.121:8780/EasyKart/Ship	29851	1798	199	2441	25413

Page 1 of 6 Displaying 1 - 10 of 60

Figure 3.89: The detailed diagnosis of the Slow page views measure

Tolerating page views

- The top-5 tolerating page views, in terms of the page load time;

Details of Tolerated Page Request

USER EXPERIENCE	REQUEST DATE	URL	PAGE LOAD TIME (MS)	BROWSER TIME (MS)	NETWORK TIME (MS)	CONTENT DOWNLOAD TIME (MS)	SERVER TIME (MS)
Slow	Aug 17, 2016 07:19:11 EDT	http://192.168.11.121:8780/EasyKart/BrowseProducts.jsp	15887	4614	192	5639	5442
Slow	Aug 17, 2016 07:04:18 EDT	http://192.168.11.121:8780/EasyKart/BrowseProducts.jsp	15887	4614	192	5639	5442
Slow	Aug 17, 2016 07:19:11 EDT	http://192.168.11.121:8780/EasyKart/PaymentPage.jsp	15774	1058	179	1293	13244
Slow	Aug 17, 2016 07:04:18 EDT	http://192.168.11.121:8780/EasyKart/PaymentPage.jsp	15774	1058	179	1293	13244
Slow	Aug 17, 2016 07:19:11 EDT	http://192.168.11.121:8780/EasyKart/PaymentPage.jsp	15614	1044	221	1277	13072
Slow	Aug 17, 2016 07:04:18 EDT	http://192.168.11.121:8780/EasyKart/PaymentPage.jsp	15614	1044	221	1277	13072
Slow	Aug 17, 2016 07:19:11	http://192.168.11.121:8780/EasyKart/Brows	15488	4491	209	5490	5298

Page 1 of 2 Displaying 1 - 10 of 16

Figure 3.90: The detailed diagnosis of the Tolerating page views measure

Frustrated page views:

- The top-5 frustrated page views, in terms of the page load time;

Details of Frustrated Page Request

USER EXPERIENCE	REQUEST DATE	URL	PAGE LOAD TIME (MS)	BROWSER TIME (MS)	NETWORK TIME (MS)	CONTENT DOWNLOAD TIME (MS)	SERVER TIME (MS)
Slow	Aug 18, 2016 01:44:50 EDT	http://192.168.11.121:8780/EasyKart/PaymentPage.jsp	30126	1816	201	2462	25647
Slow	Aug 18, 2016 02:29:56 EDT	http://192.168.11.121:8780/EasyKart/PaymentPage.jsp	30126	1816	201	2462	25647
Slow	Aug 18, 2016 01:44:50 EDT	http://192.168.11.121:8780/EasyKart/PaymentPage.jsp	30017	1808	201	2454	25554
Slow	Aug 18, 2016 02:29:56 EDT	http://192.168.11.121:8780/EasyKart/PaymentPage.jsp	30017	1808	201	2454	25554
Slow	Aug 18, 2016 01:44:50 EDT	http://192.168.11.121:8780/EasyKart/PaymentPage.jsp	29890	1801	199	2443	25447
Slow	Aug 18, 2016 02:29:56 EDT	http://192.168.11.121:8780/EasyKart/PaymentPage.jsp	29890	1801	199	2443	25447
Slow	Aug 18, 2016 01:54:50	http://192.168.11.121:8780/EasyKart/Ship	29851	1798	199	2441	25413

Page 1 of 6 Displaying 1 - 10 of 60

Figure 3.91: The detailed diagnosis of the Frustrated page views measure

Satisfied page views

- The top-5 page views accessed recently, which registered a satisfactory user experience – a satisfactory user experience is one where the *Average page load time* of a page view falls well below the **SLOW TRANSACTION CUTOFF** configured for the **Web site** test. **Note that such page views can have JavaScript errors**.

Details of Satisfied Page Request

USER EXPERIENCE	REQUEST DATE	URL	PAGE LOAD TIME (MS)	BROWSER TIME (MS)	NETWORK TIME (MS)	CONTENT DOWNLOAD TIME (MS)	SERVER TIME (MS)
Healthy	Aug 18, 2016 02:04:50 EDT	http://192.168.11.121:8780/EasyKart/AddToCart.jsp	916	55	168	75	618
Healthy	Aug 18, 2016 02:04:50 EDT	http://192.168.11.121:8780/EasyKart/AddToCart.jsp	802	49	168	65	520
Healthy	Aug 18, 2016 01:54:50 EDT	http://192.168.11.121:8780/EasyKart/AddToCart.jsp	761	46	167	62	486
Healthy	Aug 18, 2016 01:54:50 EDT	http://192.168.11.121:8780/EasyKart/ShippingPage.jsp	431	26	167	35	203
Healthy	Aug 18, 2016 01:44:50 EDT	http://192.168.11.121:8780/EasyKart/AddToCart.jsp	235	14	167	19	35
Healthy	Aug 18, 2016 02:29:56 EDT	http://192.168.11.121:8780/EasyKart/AddToCart.jsp	235	14	167	19	35

Page 1 of 1 Displaying 1 - 8 of 8

Figure 3.92: The detailed diagnosis of the Satisfied page views measure

Desktop page views

- The list of page views, out of the top-5 slow, error, and healthy page views that occurred recently, which were launched from desktops

Details of Desktop Page Request

USER EXPERIENCE	REQUEST DATE	URL	PAGE LOAD TIME (MS)	BROWSER TIME (MS)	NETWORK TIME (MS)	CONTENT DOWNLOAD TIME (MS)	SERVER TIME (MS)
Slow	Aug 18, 2016 01:44:50 EDT	http://192.168.11.121:8780/EasyKart/PaymentPage.jsp	30126	1816	201	2462	25647
Slow	Aug 18, 2016 02:29:56 EDT	http://192.168.11.121:8780/EasyKart/PaymentPage.jsp	30126	1816	201	2462	25647
Slow	Aug 18, 2016 01:44:50 EDT	http://192.168.11.121:8780/EasyKart/PaymentPage.jsp	30017	1808	201	2454	25554
Slow	Aug 18, 2016 02:29:56 EDT	http://192.168.11.121:8780/EasyKart/PaymentPage.jsp	30017	1808	201	2454	25554
Slow	Aug 18, 2016 01:44:50 EDT	http://192.168.11.121:8780/EasyKart/PaymentPage.jsp	29890	1801	199	2443	25447
Slow	Aug 18, 2016 02:29:56 EDT	http://192.168.11.121:8780/EasyKart/PaymentPage.jsp	29890	1801	199	2443	25447
Slow	Aug 18, 2016 01:44:50	http://192.168.11.121:8780/EasyKart/Brow	29548	1780	199	2416	25153

Page 1 of 2 Displaying 1 - 10 of 20

Figure 3.93: The detailed diagnosis of the Desktop page views measure

Mobile page views

- The list of page views, out of the top-5 slow, error, and healthy page views that occurred recently, which were launched from mobile phones

Details of Mobile Page Request

USER EXPERIENCE	REQUEST DATE	URL	PAGE LOAD TIME (MS)	BROWSER TIME (MS)	NETWORK TIME (MS)	CONTENT DOWNLOAD TIME (MS)	SERVER TIME (MS)
Slow	Aug 18, 2016 01:54:50 EDT	http://192.168.11.121:8780/EasyKart/AddToCart.jsp	29571	1781	199	2418	25173
Slow	Aug 18, 2016 02:04:50 EDT	http://192.168.11.121:8780/EasyKart/AddToCart.jsp	29528	1779	199	2414	25136
Slow	Aug 18, 2016 01:44:50 EDT	http://192.168.11.121:8780/EasyKart/AddToCart.jsp	29459	1775	199	2408	25077
Slow	Aug 18, 2016 02:29:56 EDT	http://192.168.11.121:8780/EasyKart/AddToCart.jsp	29459	1775	199	2408	25077
Slow	Aug 18, 2016 02:09:50 EDT	http://192.168.11.121:8780/EasyKart/AddToCart.jsp	29429	1773	199	2406	25051
Slow	Aug 18, 2016 01:39:50 EDT	http://192.168.11.121:8780/EasyKart/AddToCart.jsp	29105	1754	199	2379	24773
Slow	Aug 18, 2016 02:24:54	http://192.168.11.121:8780/EasyKart/Add	29105	1754	199	2379	24773

Page 1 of 5 Displaying 1 - 10 of 47

Figure 3.94: The detailed diagnosis of the Mobile page views measure

Tablet page views

- The list of page views, out of the top-5 slow, error, and healthy page views that occurred recently, which were launched from tablets

Details of Tablet Page Request

USER EXPERIENCE	REQUEST DATE	URL	PAGE LOAD TIME (MS)	BROWSER TIME (MS)	NETWORK TIME (MS)	CONTENT DOWNLOAD TIME (MS)	SERVER TIME (MS)
Slow	Aug 18, 2016 01:54:50 EDT	http://192.168.11.121:8780/EasyKart/ShippingPage.jsp	29851	1798	199	2441	25413

Figure 3.95: The detailed diagnosis of the Tablet page views measure

Note:

The other tests run by the eG Real User Monitor use the same detailed diagnosis information that the Web site test collects and stores in the eG database. These tests however, pick and present in their detailed diagnosis, only that information that is relevant to their purpose. For example, the Page views measure of the Chrome descriptor of the Browsers test, by default, will scan the top-5 slow, error, and healthy transactions that occurred recently on the web site/web application, pick only those transactions that were launched from the Chrome browser, and will list all those transactions in its detailed diagnosis.

By default, the detailed diagnosis is sorted by Request date. If required, you can sort the same by **Page load time** by selecting this option from the **Sort by** drop-down in the **Detailed Diagnosis** page.

You can even zoom into a particular page by clicking on its URL in the detailed diagnosis page. The **RUM Transaction Details** page will then appear. This page graphically represents the entire path of the page

request / transaction, from the time the browser received the request to the time the server responded to it by serving the requested page. The time spent by the request at each step is also displayed in the graphic. A pie chart is also provided alongside that depicts what percentage of the page load time was spent at the browser, the network, downloading content, and the server. From the size of the slices in the pie, you can quickly and accurately locate where your request was delayed.

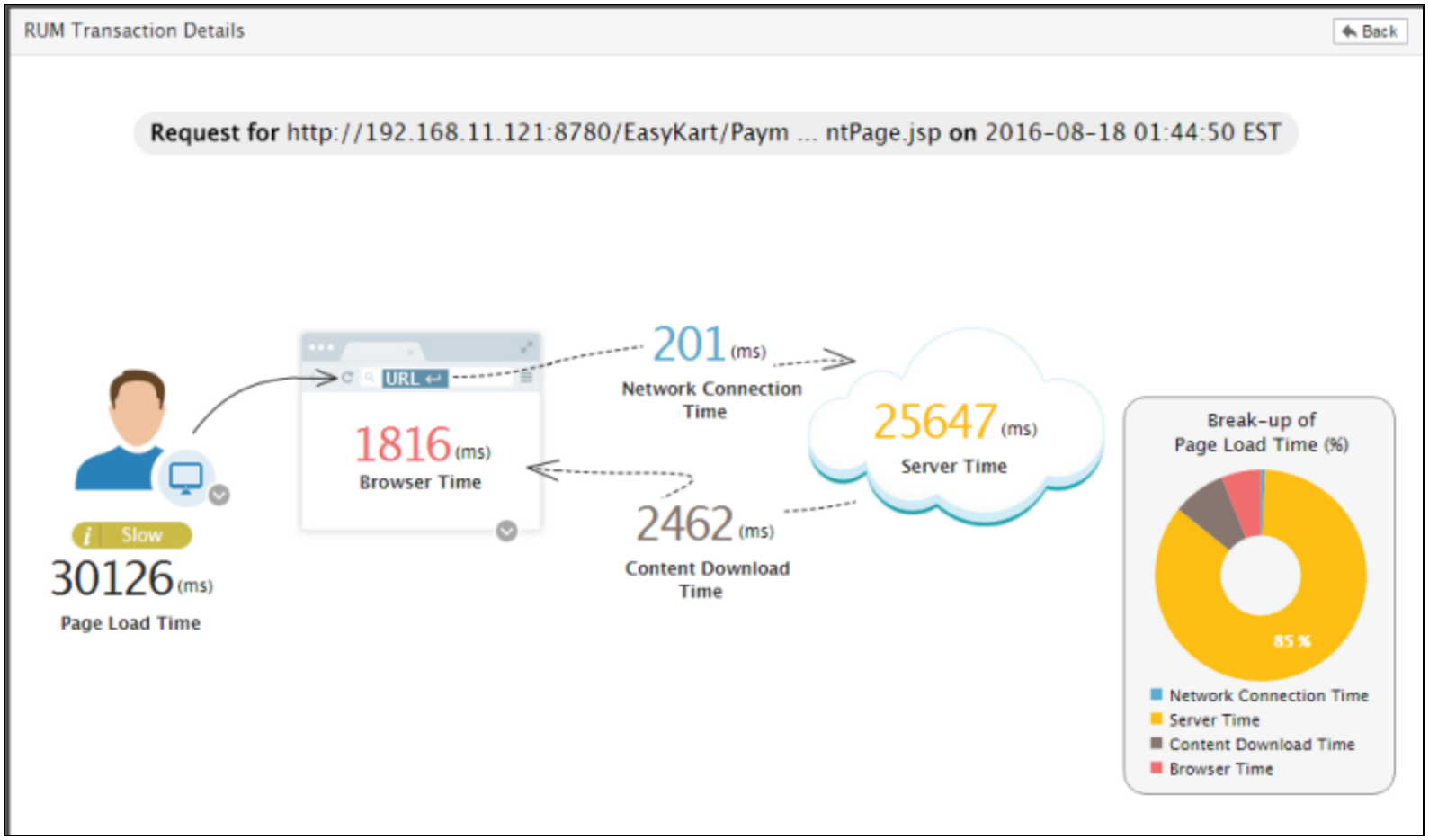


Figure 3.96: The RUM Transaction Details page displaying the entire flow of the page request/transaction

If the transaction flow diagram reveals that a latent network is what is causing the transaction to slow down, then you can easily determine why the network is latent by moving your mouse pointer over **Network Connection Time** in the diagram. A break-up of the Network connection time will then pop up, indicating exactly when network connection was bottlenecked - is it when performing a domain lookup to access the web site/web application? is it when establishing a TCP connection with the server? or was there a delay when the request was redirected to another URL?

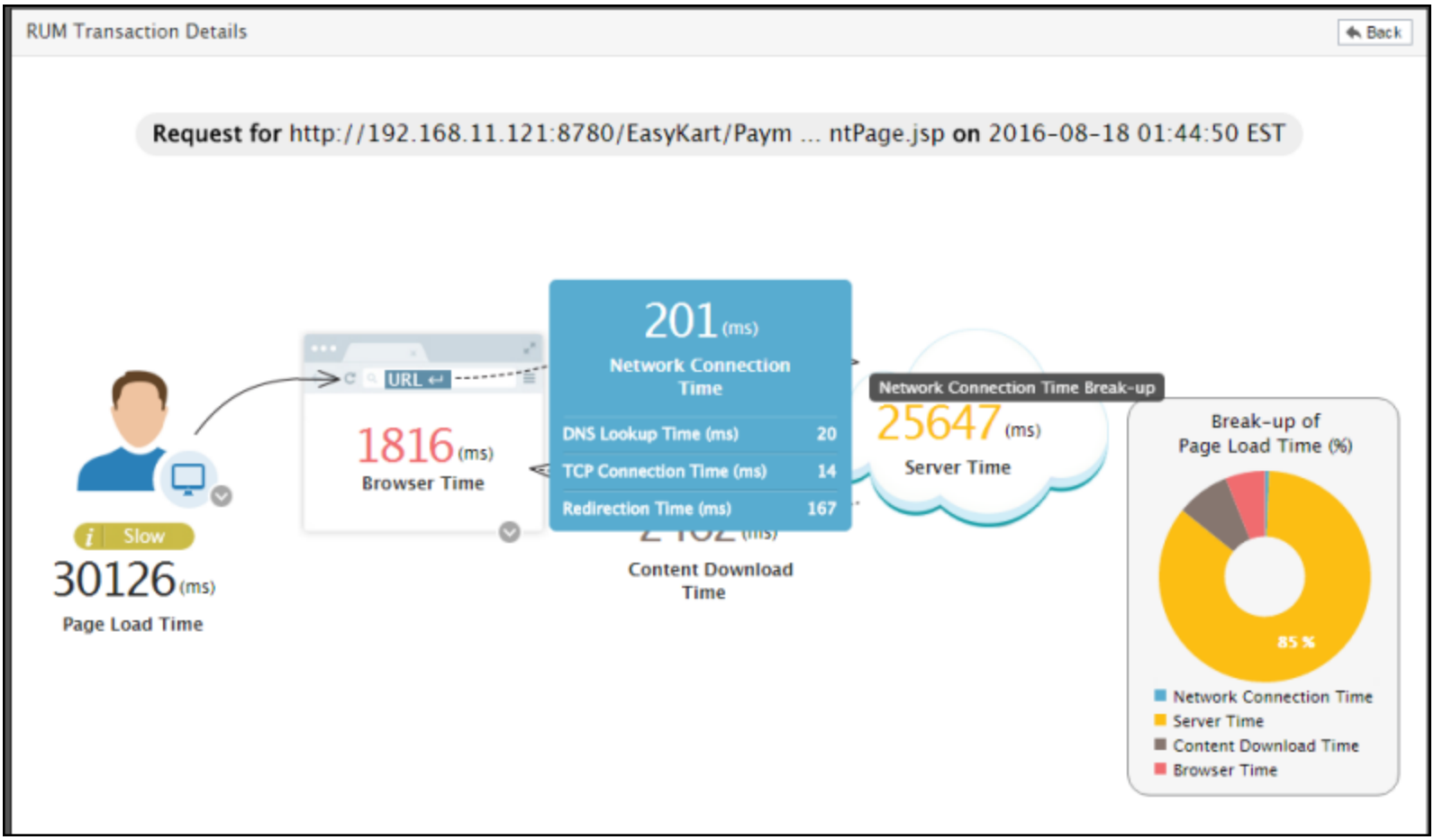


Figure 3.97: The transaction flow indicating the reason why Network connection time is high

Similarly, if a delay in content downloading caused the transaction to slow down, then you can move your mouse pointer over **Content Download Time** in the flow diagram to determine where exactly content downloading was bottlenecked - during document downloading? or during document processing?

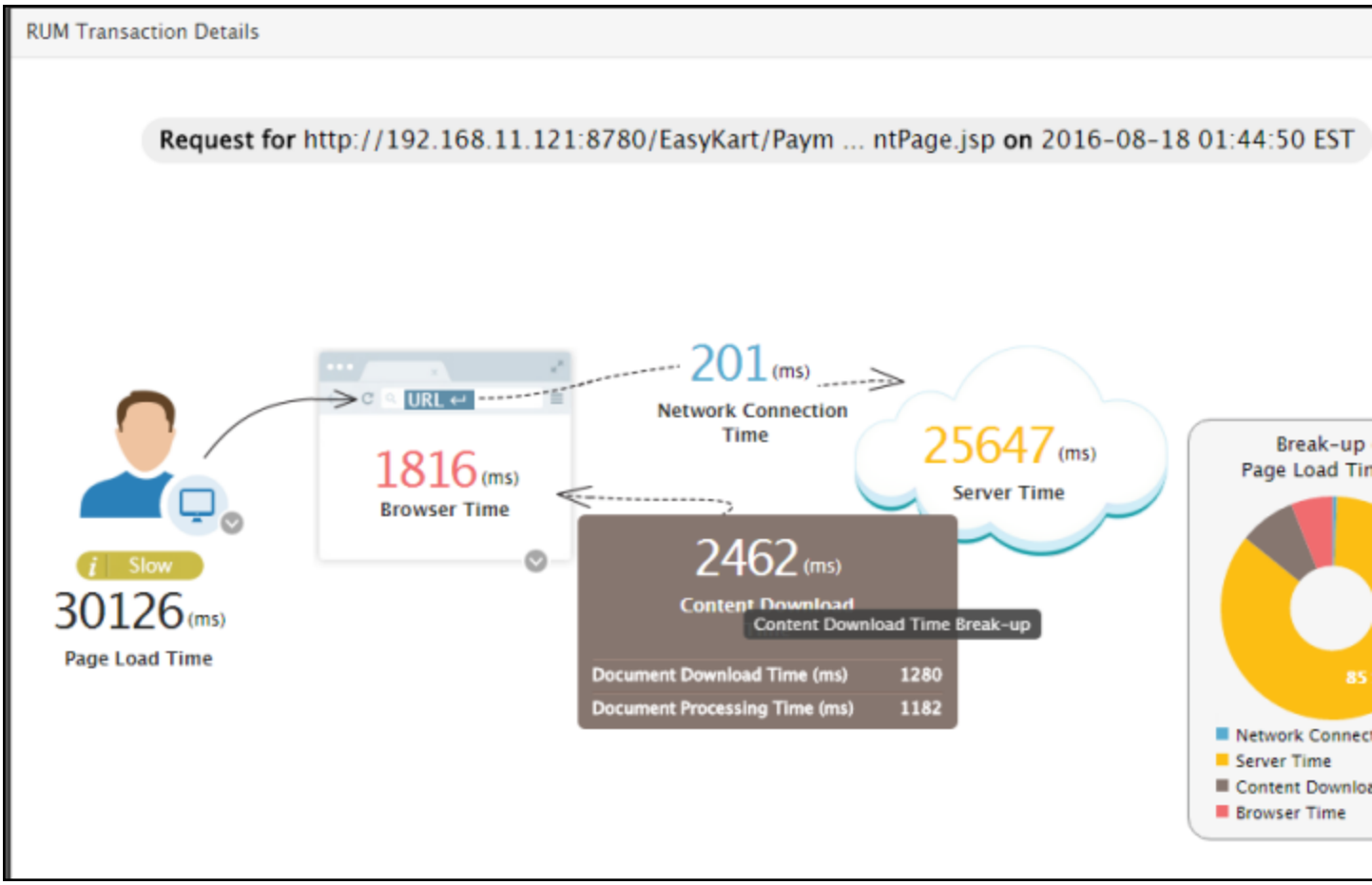


Figure 3.98: The transaction flow chart displaying the break-up of Content Download Time

The transaction flow diagram also provides you with useful user and browser information. To know the location of the user and the client from which he/she is connecting, click on the 'down arrow' adjacent to the user icon in Figure 1.15. Details such as the user location, client IP, and client OS will then be revealed (see Figure 1.15).

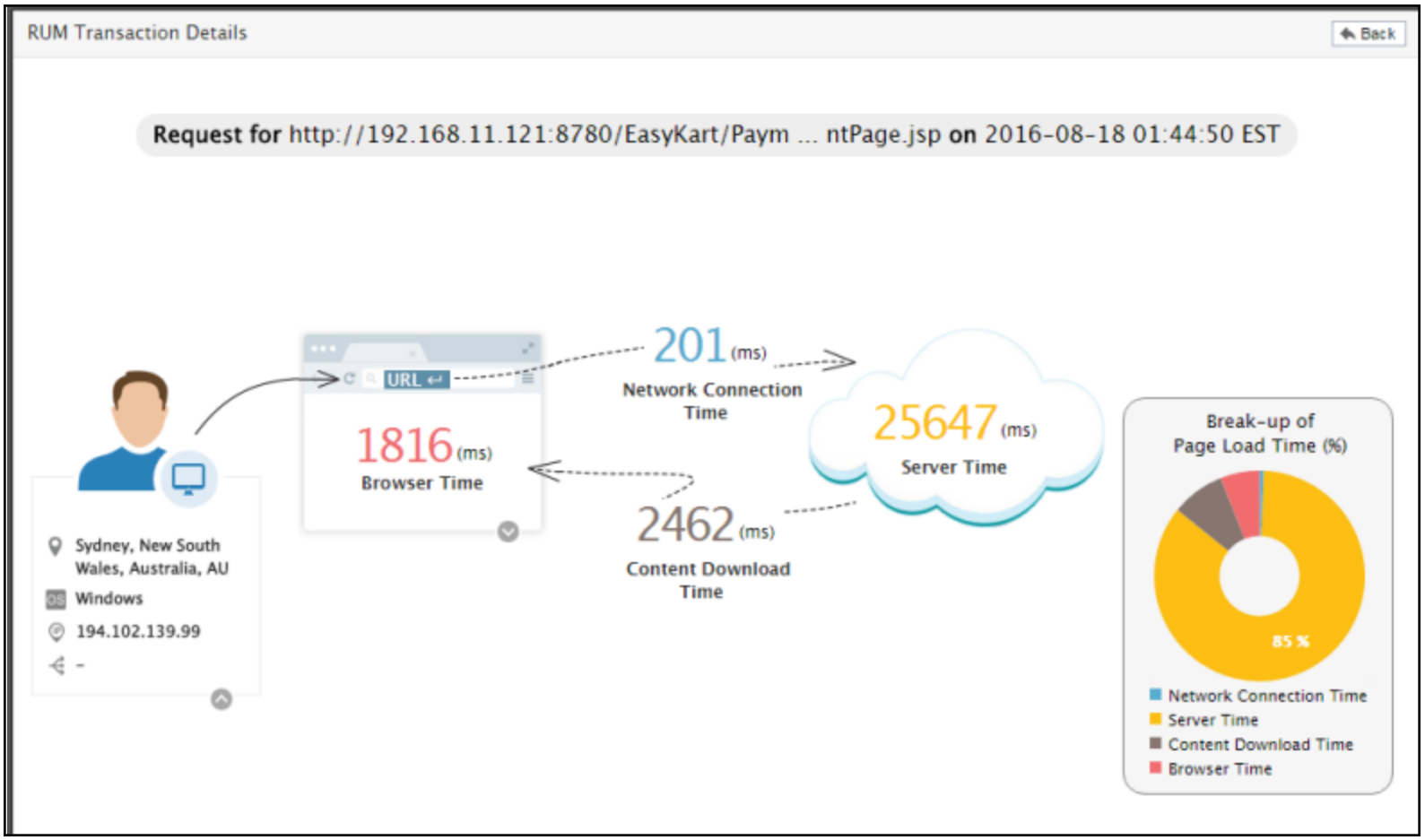


Figure 3.99: The transaction flow displaying user information

Similarly, click the down arrow adjacent to Browser Time in the transaction flow diagram to view browser information

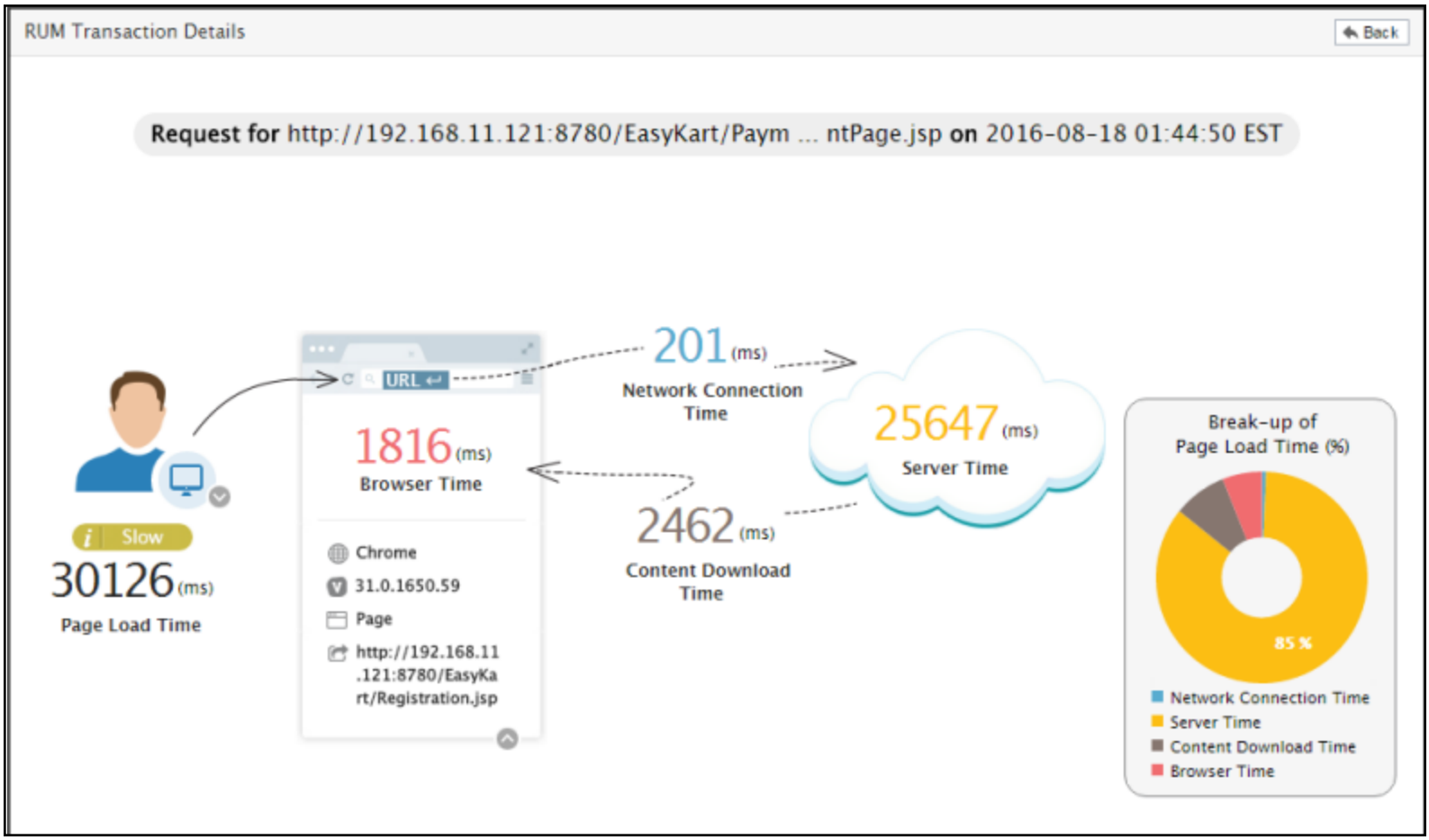


Figure 3.100: The transaction flow displaying browser information

3.6.8 The RUM Dashboard

A picture speaks louder than a thousand words! Messages conveyed using a visual medium are always more impactful and reaches the audience more quickly. This is the philosophy that drives eG's **Real User Monitor Dashboard** (see Figure 3.101). With the help of a bevy of visual tools – eg., intuitive icons, color-coded values, miniature graphs, geo maps, etc. - this real-time dashboard helps administrators understand, from just a glance, the following:

- How is the traffic to the managed web sites/web applications in your environment? Is the traffic to any web site/web application suspiciously high or low?
- From which devices is this traffic coming from – desktops? Mobile phones? Or tablets?
- Is the user experience with any managed web service poor presently? If so, what is causing service quality to degrade – delay in page loading? Or JavaScript errors?
- How many unique sessions were registered on the managed web sites/web applications to view the web pages?

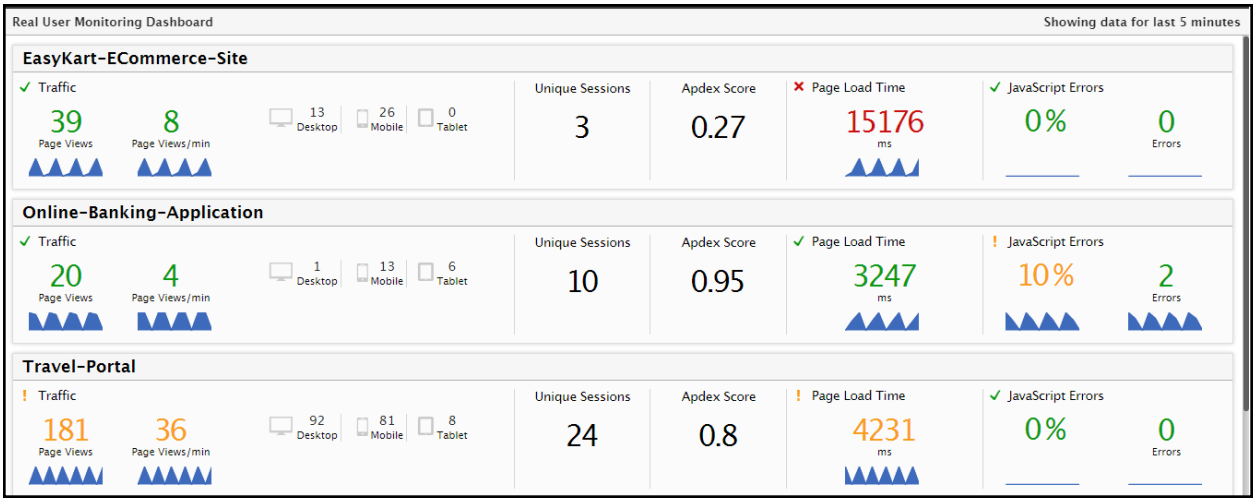


Figure 3.101: The Real User Monitor Dashboard

This way, the dashboard facilitates the rapid identification of web sites/web applications in your environment that do not enjoy user confidence and the probable reasons for this unfavorable user perception.

Moreover, the exact pages that are slow, the reason for the slowness of those pages, and the JavaScript errors that are wrecking user experience, can all be identified with a quick click from this dashboard. For instance, you can click on the *Page Load Time* measure representation in the dashboard to know which page views are experiencing slowness and why – is it because of the front end? network? or backend? (see Figure 3.102)

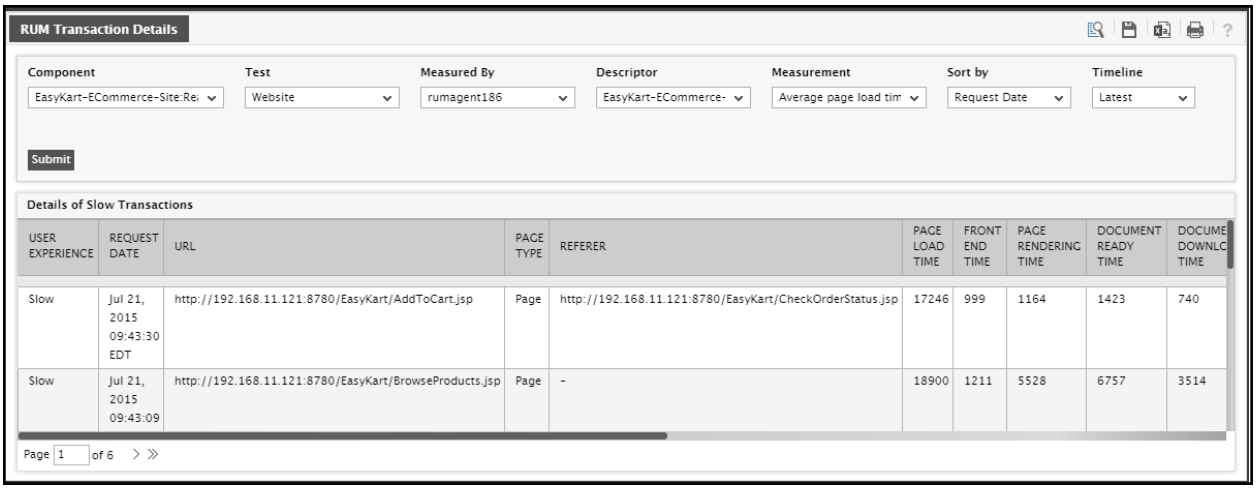


Figure 3.102: Details of slow page views where the URLs of the pages are provided along with the breakup of each page

This way, the **Real User Monitor Dashboard** takes you from the problem symptoms to the problem source in no time!

Also, drilling down from the name of the managed web site/web application in the dashboard will lead you to Figure 3.103, where you will find various perspectives to user experience represented visually.

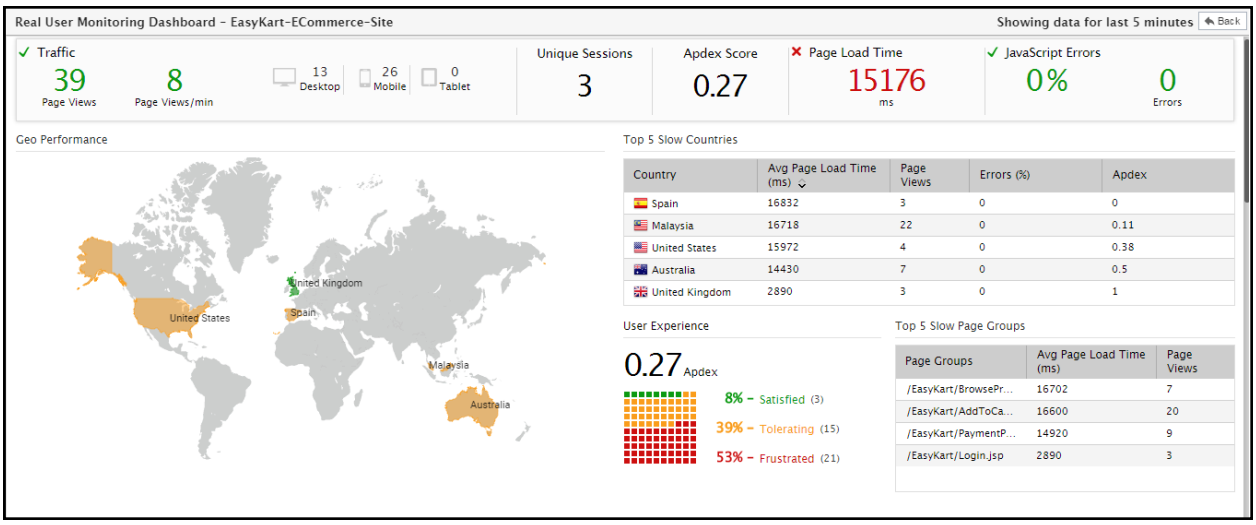


Figure 3.103: The RUM dashboard revealing the user experience per geography

Without requiring you to pour over tens of metrics, the **Geo Performance** map in Figure 3.103 quickly tells you where your users are. The **Unique Sessions** section reveals how many unique users are currently viewing the web site/web application in your environment. Also, by color-coding the countries on the basis of the experience of the users in those countries, the map intelligently differentiates between the “happy” regions and the “unhappy” ones. But, why are some regions “unhappy” with the web site/web application

performance? To know the answer, hover your mouse over each country in the map. A box depicted by Figure 3.104 will appear revealing what the users from that country experience when interacting with the target web site/web application. The page views from the country, the average page load time, and the error percentage are displayed in the box. From this, you can easily infer what is ailing the experience of users from that country – slow page loading? Or JavaScript errors?

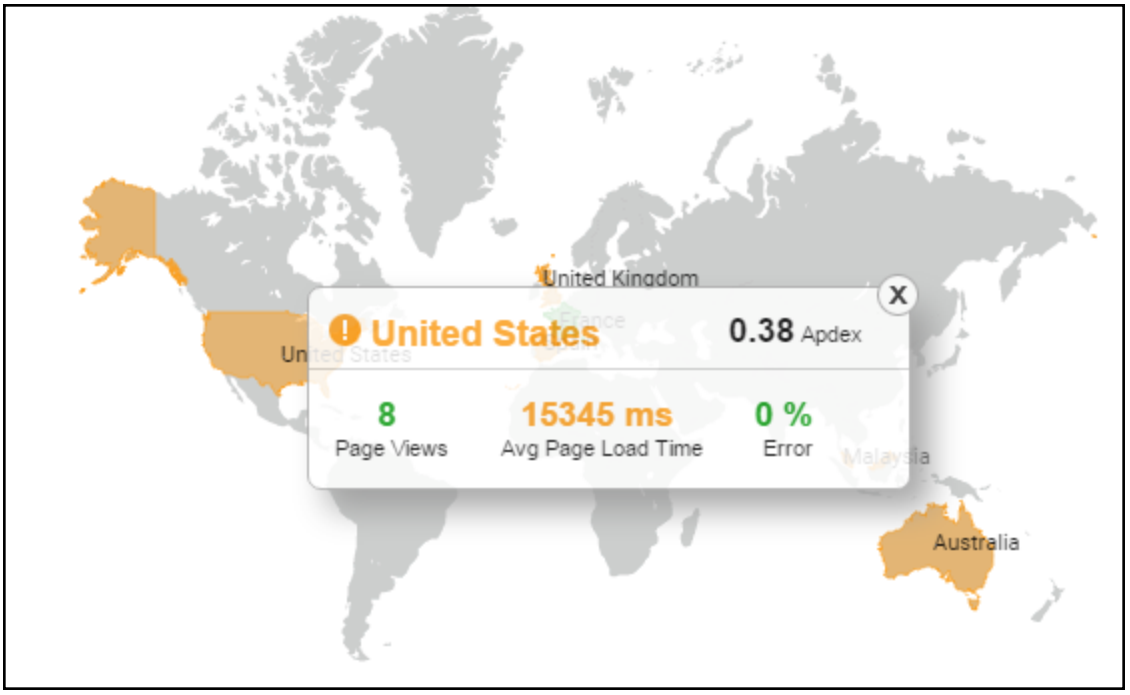


Figure 3.104: Viewing the user experience statistics of a particular country in the geo performance map

To know which of these countries is experiencing the maximum slowness, you can look at the **Top 5 Slow Countries** table to the right of the **Geo Performance** map in Figure 3.103. This table lists the top-5 slow countries, starting with that country that is seeing the maximum slowness. For each country, the number of page views from that country, the average page load time of the page views, the Apdex score, and the percentage of JavaScript errors is revealed. But the question now is, how badly is the experience of users from a few countries affecting the overall user experience with the web site/web application? Nothing reveals that better than the **Apdex Score** of the web site/web application!

The **User Experience** section of Figure 3.103 reports the current **Apdex Score** of the monitored web site/web application. If the score is very low, you can figure out why by viewing the heat map right below and the legend alongside – this reveals the percentage of satisfied, tolerating, and frustrated page views.

If the tolerating and frustrated page views are more than the satisfied page views, you will want to know where exactly user experience is suffering. The **Top 5 Slow Page Groups** section in Figure 3.103 tells you where! By listing the slowest page groups in your web site/web application, this section points you the probable problem areas – i.e., it points you to the groups that probably contain the problem pages. This way, this section takes you a step closer to isolating the real reason for a high number of tolerating and frustrated page views.

Scrolling down Figure 3.103 allows you to shift from analyzing current user experience to scrutinizing historical user experience. The first graph cross-correlates page views with the page load time registered during the last hour. From this graph (see 3.6.8), you can figure out how many of your users are being impacted by slow pages. Where users are revenue, this graph will help you understand how many users are dissatisfied with your web service, and how that dissatisfaction will impact your bottom line!

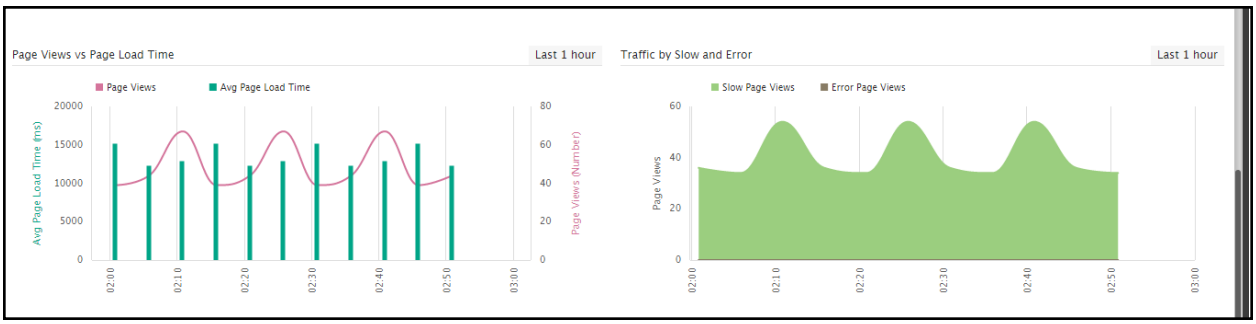


Figure 3.105: One-hour graphs cross-correlating page views with page load time and slow page views with error page views

The second graph cross-correlates slow page views with error page views during the last hour. With the help of this graph, you can determine whether/not slow page views are increasing in tandem with error page views – this could mean that JavaScript errors are probably the reason why page views are slowing down. This takes you closer to nailing the real reason for the slowness.

Scrolling down further reveals two stacked area charts (see Figure 3.106). The **Trend of Page Load Time** chart traces the high and low trends in page load time during the last hour. From this graph, you can instantly determine what has contributed the most to the increase in page load time during the last hour – the network? the backend? Or the front end? Once the problem tier is identified, you can focus your time and resources on isolating the exact problem with that tier and eliminating it, so that user experience improves rapidly. The **Trend of User Experience** graph that appears next, reveals when during the last hour the tolerating, satisfied, and frustrated page views peaked and when they were the lowest. From this graph, you can easily figure how the page viewing experience of users was at any given point in time during the last hour.

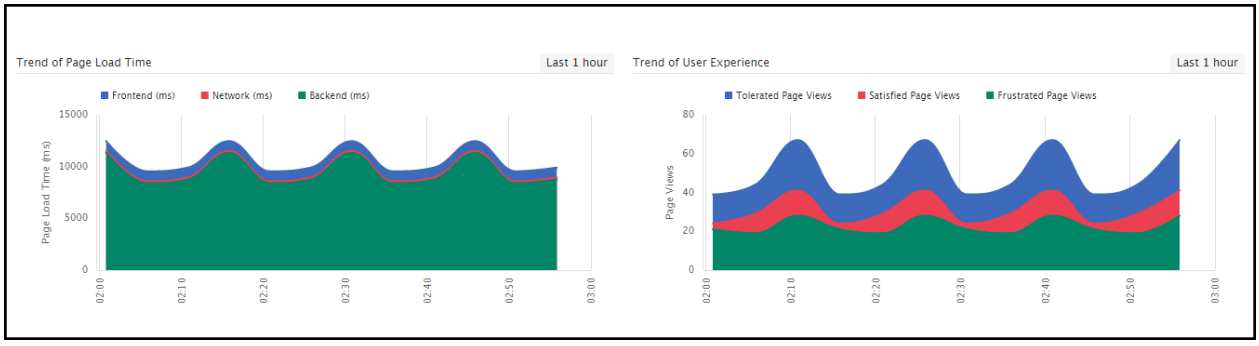


Figure 3.106: One-hour trend graphs on user experience and page load time

To know which browsers and devices are popular with your web site/web application users, use the distribution pie charts (see Figure 3.107) that will appear when you scroll down the dashboard further.

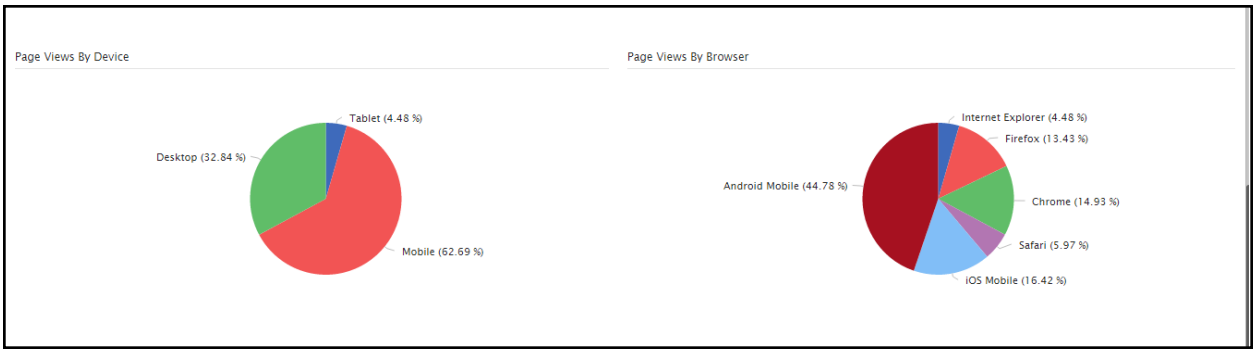


Figure 3.107: Distribution pie charts for browsers and devices

4

Troubleshooting Using the RUM Logger

To enable administrators to easily troubleshoot issues related to eG's real user monitor, eG Enterprise provides a specialized **Logger**. The logger logs key functions of the eG RUM collector and the eG agent that pulls data from the collector in various log files.

For the eG RUM collector, the logger creates the following log files in the **<EG_RUM_DATA_COLLECTOR_INSTALL_DIR>\tomcat\webapps\rumcollector\WEB-INF\rum\logs** directory (on Windows; on Unix, this will be the **/opt/rum/tomcat/webapps/rumcollector/WEB-INF/rum/logs** directory):

Log file name	Purpose
rum_collector_init.log	Tracks all initialization activities performed by the RUM collector
agent_data.log	Tracks how the RUM collector responds to requests from the eG agent
beacon_receiver.log	Tracks all activities related to transmission of beacons to the RUM collector and the writing of metrics to flat files by the collector

For the eG agent, the logger creates a **rum.log** file in the **<eg_agent_install_dir>\agent\logs directory** (on Windows; on Unix, this will be the **/opt/egurkha/agent/logs** directory). All activities related to the eG agent communication with the eG RUM collector are tracked by this log.

4.1 Configuring Logging for the eG RUM Collector

The logging level of the collector-related log files, the format of the log entries, and the maximum size up to which these log files can grow, can be custom-defined in the **logback.properties** file in the **<EG_RUM_DATA_COLLECTOR_INSTALL_DIR>\tomcat\webapps\rumcollector\web-inf\lib** directory (on Windows; on Unix, this will be the **/opt/rum/tomcat/webapps/rumcollector/WEB-INF/lib** directory).

Typically, the collector logs can be configured with one of the following log levels: *ALL* or *TRACE*, *DEBUG*, *INFO, WARN, ERROR*. Each of these log levels and their significance has been discussed below:

- *ALL* or *TRACE:* Setting the log level to *ALL* or *TRACE* enables the logger to log detailed information pertaining to the events logged. Moreover, once the log level is set to *ALL* or *TRACE*, then *DEBUG, INFO, WARN,* and *ERROR* messages will also be logged in the log files.
- *DEBUG*: Set the log level to *DEBUG* to make sure that *DEBUG* messages are logged in the log file. Using such messages, you can easily troubleshoot issues related to the operations of the RUM collector. Once the log level is set to *DEBUG*, then, *INFO, WARN,* and *ERROR* messages will also be logged in the log files. However, detailed event information provided by the *ALL* or *TRACE* level will not be available for the *DEBUG* level.

- *INFO*: To capture general information messages, set the log level to *INFO*. In this case, besides *INFO* messages, *WARN* and *ERROR* messages will also be logged in the log files.
- *ERROR*: By setting the log level to *ERROR*, you can make sure that only error messages are captured by the log files.

By default, the collector logs are configured with the *DEBUG* log level. This is why, the **LOG_LEVEL** parameter in the **logback.properties** file is set to *DEBUG* by default. Depending upon the types of events you want captured and the level of information you want logged in the log file, you can change the value of the **LOG_LEVEL** parameter to any of the levels described above.

Besides the log level, you can also configure log rolling using the **logback.properties** file. By configuring a size (in MB) against the **LOG_FILE_SIZE** parameter, you can indicate the maximum size upto which a log file can grow. Beyond that point, all entries from the log file will be automatically copied to a new log file, and all subsequent messages will be logged in the old log file. For instance, if the **LOG_FILE_SIZE** parameter is set to *5* (MB), then messages will be logged in the *agent_data* log file (for example), only until the file becomes 5 MB in size. After this point, a new log file named *agent_data1* will be created, to which all entries from the *agent_data* log will be copied. This way, by default, a maximum of 20 files will be created. This is governed by the **LOG_FILE_COUNT** parameter, with the default value *20*. You can increase or decrease the value of this parameter depending upon how many times you want the log files to roll.

You can also configure the format in which events are to be logged in the log files using the **LOG_PATTERN** parameter in the **logback.properties** file. By default, this parameter is set to the following format:

%date %-5level {%C %M} - [%L] - %msg%n

Here, *%date* refers to the date/time of the event. *%-5level* indicates the log level. *%C* indicates the class name, *%M* denotes the method name, *%L* signifies the line number, and *%msg* refers to the message.

You can change the log file pattern to remove or reposition any of the tags mentioned above.

A sample log file output is provided below:

2015-08-14 16:32:08,363 DEBUG {com.eg.RumMeasureData doPost} - [212] - Reading data from renamed service file !!! [C:\egurkha\manager\tomcat\webapps\rumcollector\WEB- INF/rum/data/For_ OOME-1439550128151]

Here:

2015-08-14 16:32:08,363 is the date/time of the event.

DEBUG is the log level

com.eg.RumMeasureData is the class name

doPost is the method

212 is the line number

Reading data from renamed service file !!! [C:\egurkha\manager\tomcat\webapps\rumcollector\WEB-INF/rum/data/For_OOME-1439550128151] is the message

4.2 Configuring Logging for the eG Agent Communicating with the eG RUM Collector

As already mentioned, the logger creates a **rum.log** file in the **<EG_AGENT_INSTALL_DIR>\agent\logs** directory (on Windows; on Unix, this will be the **/opt/egurkha/agent/logs** directory) to track the eG remote agent's communication with the eG RUM collector.

Like the collector-related logs discussed in Section **4.1** section, you can configure the properties of the logging activity of the **rum.log** as well. For this purpose, eG provides a **eg_logback.xml** file. This file is available in the **<EG_AGENT_INSTALL_DIR>\lib** directory (on Windows; on Unix, this will be the **opt/egurkha/lib** directory). The contents of this XML file are as follows:

```
<?xml version="1.0"?>
 <configuration scanPeriod="5 seconds" scan="true">
 <appender class="com.eg.ch.qos.logback.core.rolling.RollingFileAppender"
name="RUMLOGGER">
 <file>${egurkhaInstallDir}/agent/logs/rum.log</file>
 <rollingPolicy class="com.eg.ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
<fileNamePattern>${egurkhaInstallDir}/agent/logs/rum%i.log</fileNamePattern>
 <minIndex>1</minIndex>
<maxIndex>20</maxIndex>
 </rollingPolicy>
 <triggeringPolicy class="com.eg.ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
<maxFileSize>20MB</maxFileSize>
 </triggeringPolicy>
 <encoder>
<pattern>%date{dd/MM/yyyy HH:mm:ss} %level - %msg%n</pattern>
 </encoder>
 </appender>
 <logger name="RUMLOGGER" additivity="true" level="DEBUG">
 <appender-ref ref="RUMLOGGER"/>
 </logger>
 </configuration>
```

The lines/tags in **Bold** in the above extract are the ones that govern how logging is performed by the **rum.log**. Let's take a look at these **Bold** entries, in random order.

Typically, the **rum.log** can be configured with one of the following log levels: *ALL* or *TRACE, DEBUG, INFO, WARN, ERROR*. Each of these log levels and their significance has been discussed below:

- *ALL* or *TRACE:* Setting the log level to *ALL* or *TRACE* enables the logger to log detailed information pertaining to the events logged. Moreover, once the log level is set to *ALL* or *TRACE*, then *DEBUG, INFO, WARN,* and *ERROR* messages will also be logged in the log files.
- *DEBUG*: Set the log level to *DEBUG* to make sure that *DEBUG* messages are logged in the log file. Using such messages, you can easily troubleshoot issues related to the eG agent-collector communication. Once the log level is set to *DEBUG*, then, *INFO, WARN,* and *ERROR* messages will also be logged in the

log files. However, detailed event information provided by the *ALL* or *TRACE* level will not be available for the *DEBUG* level.

- *INFO*: To capture general information messages, set the log level to *INFO*. In this case, besides *INFO* messages, *WARN* and *ERROR* messages will also be logged in the log files.
- *ERROR*: By setting the log level to *ERROR*, you can make sure that only error messages are captured by the log files.

By default, the **rum.log** is configured with the *DEBUG* log level. This is why, the **level** parameter in the line that begins with <logger name="RUMLOGGER" additivity="true". . .> is set to *"DEBUG"* by default. Depending upon the types of events you want captured and the level of information you want logged in the log file, you can change the value of the **level** parameter to any of the levels described above.

Besides the log level, you can also configure log rolling using the **eg_logback.xml** file. By configuring a size (in MB) within the *<maxFileSize></maxFileSize>* XML tags, you can indicate the maximum size upto which the **rum.log** can grow. Beyond that point, all entries from the log file will be automatically copied to a new log file, and all subsequent messages will be logged in the old log file. By default, *20* MB is the maximum size upto which the **rum.log** can grow. This can be increased or decreased according to the logging needs of your environment. If you go with the default setting for instance, messages will be logged in the *rum.log* file (for example), only until the file becomes *20* MB in size. After this point, a new log file named *rum1.log* (by default) will be created, to which all entries from the *rum.log* log will be copied. This default naming convention can also be overridden by editing the value contained within the *<fileNamePattern></fileNamePattern>* tags. The default value within these tags is as follows:

${egurkhaInstallDir}/agent/logs/rum%i.log

Here, *{egurkhaInstallDir}/agent/logs* indicates the default destination for the **rum.log** file, and the *rum%i.log* indicates the pattern using which the additional **rum.log** files created in the default destination are to be named. The *%i* in the *rum%i.log* pattern, represents an integer which will be incremented for every additional log file that is created. Going by this default pattern, once the size of the **rum.log** file violates its *<maxFileSize></maxFileSize>* threshold, the next log file that is created will be named *rum1.log*, the one after that will be named *rum2.log* and so on.

In this manner, a maximum of 20 log files will be created by default. This is governed by the *<maxIndex></maxIndex>* tags in the *eg_logback.xml* file. By default, the value contained within these tags is *20*. You can increase or decrease this value to suit your needs.

You can also configure the format in which events are to be logged in the **rum.log** using the *<pattern></pattern>* tags. By default, the following pattern is configured within these tags:

%date{dd/MM/yyyy HH:mm:ss} %level - %msg%n

Here, *%date{dd/MM/yyyy HH:mm:ss}* refers to the date/time of the event. *%level* indicates the log level. *%msg* refers to the message.

You can change the log file pattern to remove or reposition any of the variables mentioned above.

A sample log file output is provided below:

2015-08-17 18:23:01,496 DEBUG - [To_Test][RUMDeviceTest] IS STARTING !!!

Here:

2015-08-17 18:23:01,496 is the date/time of the event.

DEBUG is the log level

*[To_Test][RUMDeviceTest] IS STARTING !!!*is the message

If required, you can remove one/more of these variables from the pattern specification or add more variables to it. The variables that can be added are as follows:

- *%C*, to include the class name in the log entry
- *%M*, to include the method name
- *%L*, to include the line number

For instance, your log file pattern can be as follows:

%date{dd/MM/yyyy HH:mm:ss} %level {%C %M} – [%L] - %msg%n

In this case, your log entry will be as follows:

2015-08-17 18:23:01,496 DEBUG {RUMDeviceTest computeMeasures} - [30] - [To_Test][RUMDeviceTest] IS STARTING !!!

Here:

2015-08-17 18:23:01,496 is the date/time of the event.

DEBUG is the log level

RUMDeviceTest is the class name

computeMeasures is the method name

30 is the line number

[To_Test][RUMDeviceTest] IS STARTING !!! is the message

5

Troubleshooting Failure of the eG RUM Tests

If all the eG RUM tests fail to report any metrics, you may want to do the following to determine the reason for the failure and to decide on a course of action:

1. First, check whether the web site/web application being monitored is actively accessed by users. If traffic to a web site / web application is 0, then the eG tests will not report any metrics.
2. If the web site / web application is being actively used, then proceed to check whether/not the browser downloaded the JavaScript from the RUM collector. To verify that, do the following:
   - First open the browser, and then, open the web page in which you have inserted the JavaScript.
   - Next, press F12 on your keyboard to open the Developer console. Figure 5.1 will appear.

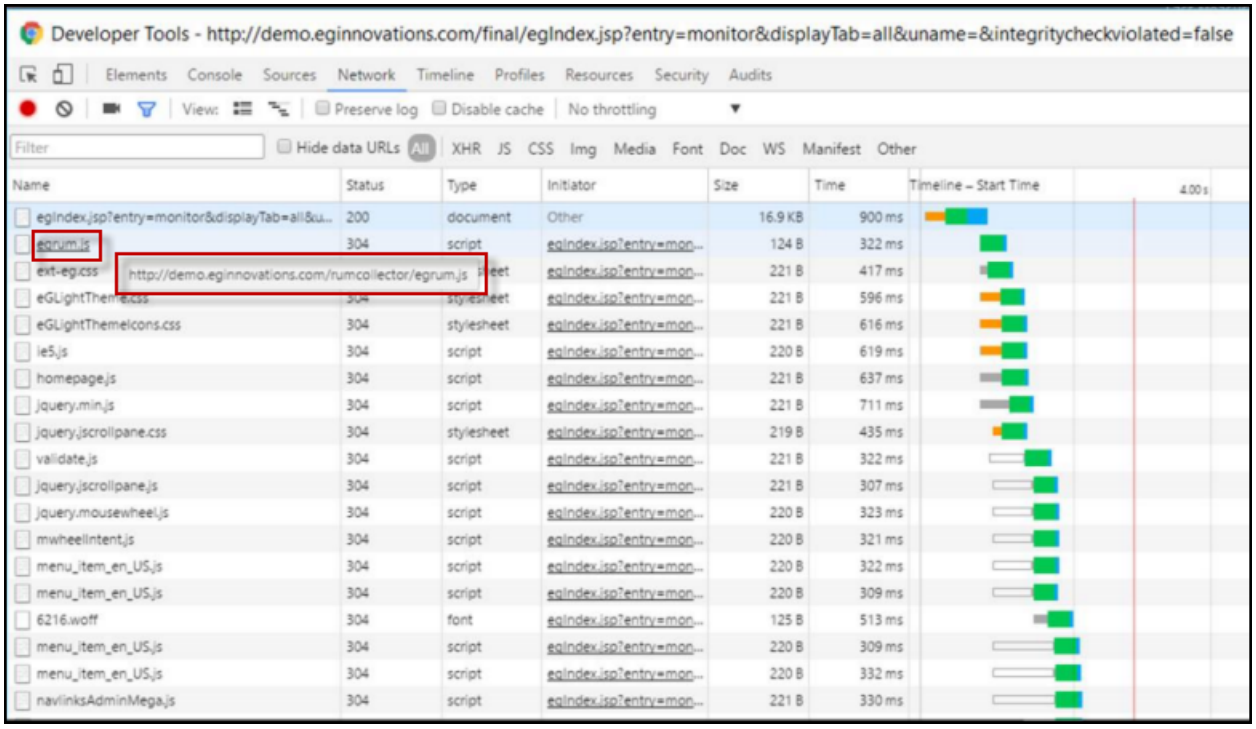


Figure 5.1: Developer Tools console revealing whether/not the JavaScript is successfully inserted in a web page

- When Figure 5.1 appears, look for **egrum.js** in it. Once you find the entry, then check the status value that corresponds to that entry in the **Status** column of Figure 5.1. If the status value is 200 or 304, then its a clear indication that the browser has successfully downloaded the JavaScript from the

RUM collector. If the status value is not 200 or 304, then it means that the browser was unable to download the JavaScript from the RUM collector. The reasons for the failure will vary according to the status value. For instance, the value 500 indicates that RUM collector was not running when the download was attempted. In such a case, starting the RUM collector can solve the problem.

- If the **egrum.js** listing is available in the Developer console, move your mouse pointer over that listing to know which RUM collector it was downloaded from. The URL that appears will be of the format: *http://<IPAddress_or_Hostname_of_ RUMCollector> : <RUMCollectorPort> /rumcollector/egrum.js*. Now, check whether the *IP:Port* in the URL is that of the RUM collector that you have configured for receiving beacons from this web site/web application. If so, it means that the **egrum.js** has been downloaded from the right RUM collector only.

3. Once you have confirmed that the JavaScript has been downloaded successfully from the correct RUM collector, then proceed.

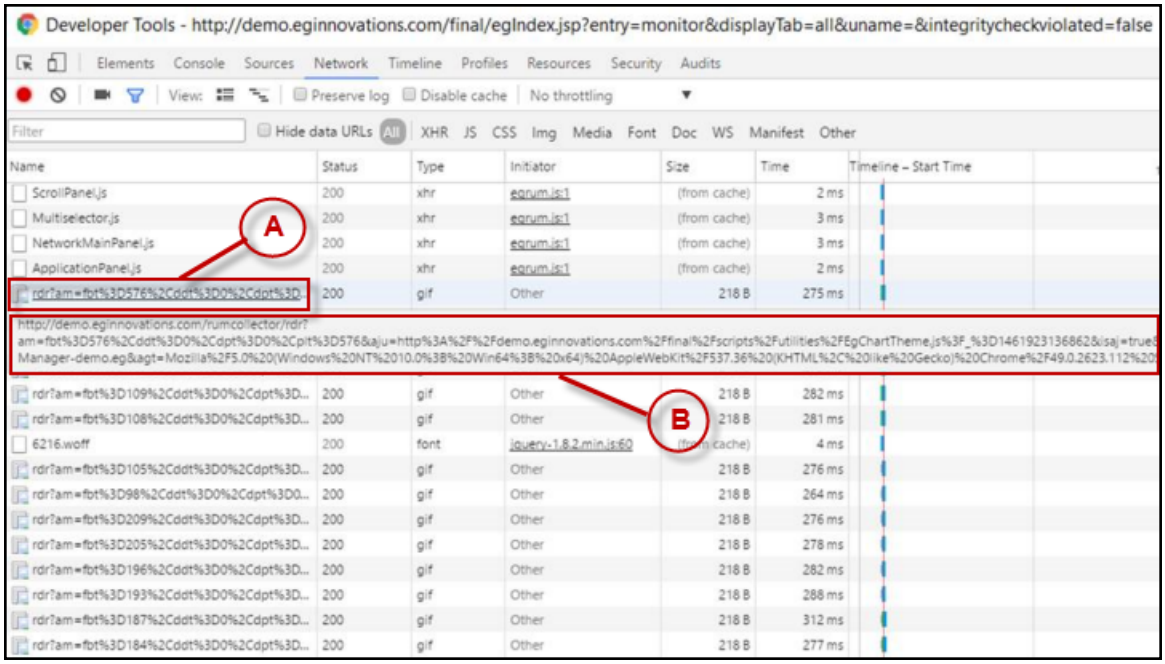


Figure 5.2: Determining whether/not the browser is sending performance beacons to the RUM collector

When Figure 5.2 appears, do the following:

- Look for the entry indicated by 'A' in Figure 5.2. If the entry appears, it implies that the browser has attempted to send a beacon to the RUM collector.
- Then, check the value displayed in the **Status** column of entry 'A'. If the value is 200, it denotes that the beacon transmission was successful. Any other status value is indicative of a transmission failure. The real reason for the failure can be ascertained from the exact status value displayed. For instance, if the status value reported is 500, it means that the RUM collector was not running when the beacon was sent to it. In such a case, starting the RUM collector will solve the problem.
- If beacon transmission was successful, then move your mouse pointer over the entry 'A' in Figure 5.2. This will reveal the URL to which the beacon was sent. In the example of Figure 5.2, this is the

URL that is indicated by 'B'. This URL will be of the format: *http://<IPAddress_or_Hostname_of_RUMCollector>:<RUMCollectorPort>/rumcollector/<Details_contained_in_the_beacon>*. To check whether the beacons were sent to the correct RUM collector, take a closer look at the *<IPAddress_or_ Hostname_ of RUMCollector>:<RUMCollectorPort>* in the URL. If this is that of the RUM collector that you have configured for receiving beacons from this web site/web application, then you can be rest assured that the beacons were sent to the right collector only.

4. Now, if all the checks above are positive, you can be certain that the browser has sent the performance beacons to the collector. If this is the case, then proceed to check if the problem is at the RUM collector end - in other words, the collector may not have been able to write the beacons into its data folder. To verify that, do the following:
   - Login to the RUM collector.
   - If the RUM collector that is bundled with the eG manager is the one in use in your environment, then open the following folder on the collector host: **<EG_ RUM_ COLLECTOR_ INSTALL_ DIR> \manager\tomcat\wenapps\WEB- INF\rum\data** (on Windows; on Unix, this will be **/opt/egurkha/manager/tomcat/webapps/WEB- INF/rum/data**). If the RUM collector has been installed on a stand- alone host, then open the following folder on the host: **<EG_ RUM_ COLLECTOR_INSTALL_DIR>\rum\tomcat\wenapps\WEB-INF\rum\data** (on Windows; on Unix, this will be **/opt/egurkha/rum/tomcat/webapps/WEB-INF/rum/data**). Once you open the folder, refresh the folder to see if any new data files are created in the folder. If they are created, then it clearly indicates that the collector has received the beacons. On the other hand, if no new data files are created in the folder even after refreshing it, it could mean that the collector was not able to write the beacons into the data folder.
   - To know what could have caused the write failure, use the **beacon_receiver.log** file in the **<EG_ RUM_ DATA_ COLLECTOR_ INSTALL_ DIR >\tomcat\webapps\rumcollector\ WEB-INF \rum\logs** directory (on Windows; on Unix, this will be the **/opt/rum/tomcat/webapps/rumcollector/WEB-INF/rum/logs** directory). This file logs all activities related to transmission of beacons to the RUM collector and the writing of metrics to flat files by the collector. To know how to configure this log, click on this link: Configuring Logging for the eG RUM Collector. You should be able to easily troubleshoot issues in beacon transmission and writing using this log file.
5. If flat files are created in the data folder, but the tests still fail, it only means that the eG agent running the tests was unable to read from the flat files. Errors in communication between the agent and the RUM collector could have lead to this problem. This issue could also occur if the agent stops running suddenly, or was not started at all. To determine why the agent failed to read from the files, use the **rum.log** file available in the **<EG_AGENT_INSTALLIDIRr>\agent\logs** directory (on Windows; on Unix, this will be the **/opt/egurkha/agent/logs** directory). All activities related to the eG agent communication with the eG RUM collector are tracked by this log. To know how to configure this log, click on this link: Configuring Logging for the eG Agent Communicating with the eG RUM Collector. You should be able to easily troubleshoot issues in agent-collector communication using this log file.

6

The eG Real User Monitor FAQ

This topic provides answers to some common questions that users raise when using eG RUM. You will also find some tips on fine-tuning eG's RUM.

6.1 Introduction to eG RUM

1. **What does the eG Real User Monitor do?**

   eG's Real User Monitor tracks LIVE, the transactions of real users (not emulated) to web sites/web applications and measures the response time of each transaction in real-time. In the process, eG RUM pinpoints slow transactions and accurately isolates the reason for the slowness – is it the network? front end? or backend?

2. **How does the eG RUM work?**

   - A small Java script is inserted in every page to be monitored in the target web site/web application.
   - Whenever the browser loads one of these web pages in response to a user request, the script runs and collects response time metrics.
   - The browser sends performance beacons carrying the gathered metrics to a software component called the **eG RUM collector**.
   - The collector receives the metrics and stores them locally in flat files.
   - An eG remote agent then polls the RUM collector at regular intervals to collect the request and response time metrics.
   - The remote eG agent then aggregates the collected metrics, isolates deviations, and transmits both the aggregated results and deviations to the eG manager.
   - The eG manager presents the performance and problem information in the eG monitoring console and also stores the same in the eG backend.

3. **What type of web sites can be monitored by eG RUM?**

   Broadly, eG RUM supports the Java, .Net, and PHP frameworks. In addition, any web page/web application containing HTML content, where a JavaScript code can be included in the <*head*> tag, can be monitored using eG RUM.

6.2 eG RUM Licensing

1. **How is eG RUM licensed?**

   Every *Real User Monitor* component that is managed using the eG admin interface will consume a **Premium Monitor** license. The collectors however are not licensed, and neither are the remote agents.

2. **I have a load balanced environment where there are multiple IIS servers serving the same website. From a licensing perspective, will I need ONE license or MULTIPLE licenses?**

   Regardless of number of IIS servers that serve the website, you will need to purchase only one premium license per website.

3. **How many eG monitor licenses will I require if I monitor two sub-websites as a single, unique site?**

   Since both web sites are being monitored as a single, unique site, both sites will send the same SiteId in the beacon.

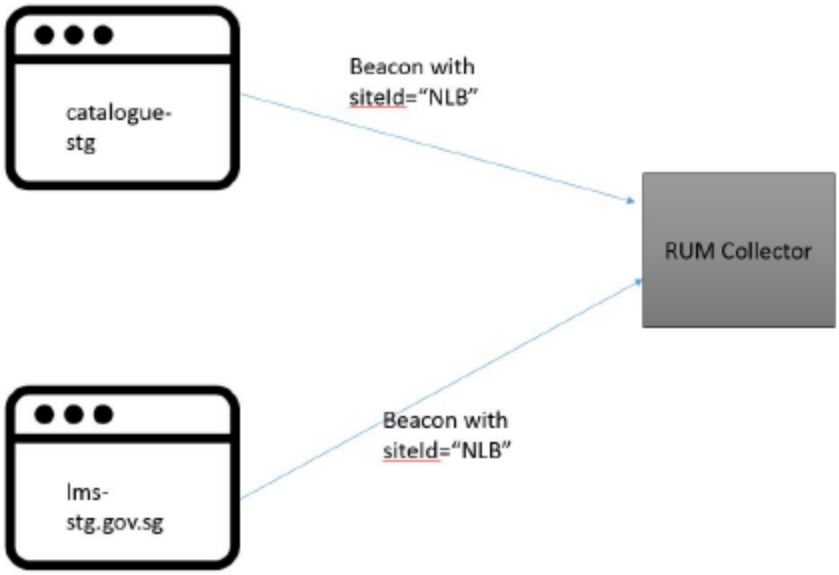


Figure 6.1: Two web sites sending beacons to the RUM collector using a single Site ID

   Typically, each SiteID maps to a RUM component in eG. As only a single SiteID is used in the beacon, the metrics for both sites will be captured by only ONE RUM component. This RUM component will consume ONE PREMIUM MONITOR license only.

4. **In the example above, can I view the two sub-websites as two different web applications in the Real User Monitoring dashboard, without any additional licenses?**

   No. With a single Premium monitor license, you cannot monitor the health of two web sites. For that, you will have to manage the sub-websites as two different RUM components in eG, thereby consuming

two premium monitor licenses. If this is done, then each of the web sites will send a different SiteID in the beacon (see Figure 1.20).

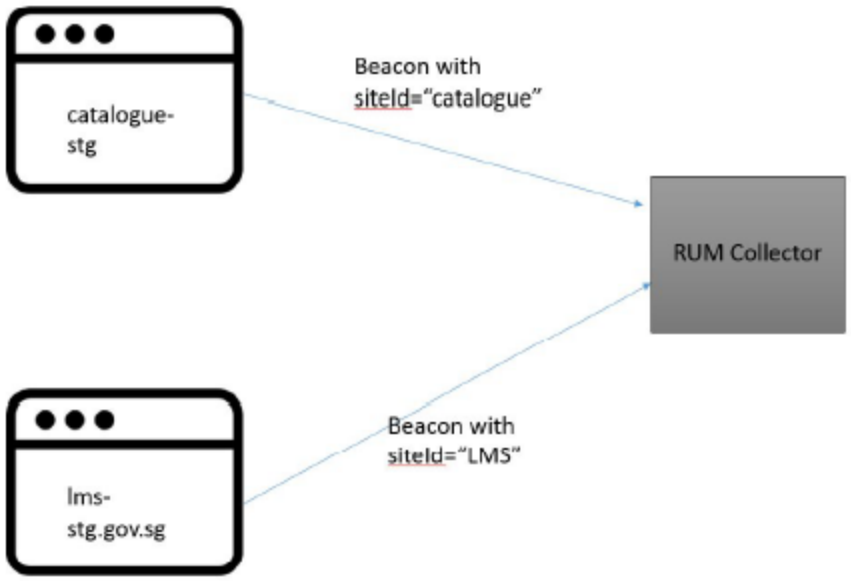


Figure 6.2: Two web sites sending beacons to the RUM collector using different Site IDs

Both these websites will then be available as distinct web applications in the dashboard.

5. **Assume that two IIS web servers - one in the internal network and the other in a DMZ - are together hosting a single web site. To monitor this web site, how many monitor licenses and RUM collectors are required?**

   The count of licenses and RUM collectors required for web site monitoring is not governed by the number of IIS web servers hosting that web site. In the case of your environment therefore, to monitor a single web site, you will only need to manage it as a single RUM component in eG, which in turn will consume only one premium monitor license.

6.3 The eG RUM Collector

1. **What is the eG RUM collector?**

   The eG RUM collector is a Tomcat-based software component that serves as an interface between the web site/web application being monitored and the eG agent. The collector has been specially designed to receive user experience metrics related to a target web site/web application from user browsers at configured intervals, and store them locally until such time the eG remote agent requests for them.

2. **Can RUM collector be deployed on a Solaris or Linux-based operating system?**

Technically, you can install RUM Collector on Solaris or Linux. However, we do not have a dedicated installer for any Unix flavor. Therefore, you would have to install Tomcat separately on the above said Unix environments and then deploy the RUMCollector application (war file) on Tomcat., do the following:

3. **Can the RUM collector discover the IP address of the browser client used by the user? If so, how do I configure this discovery?**

   Yes. The eG RUM collector is capable of discovering the IP address of the browser client of the user. To enable this discovery, follow the steps below:

   - Edit the **rum.properties** file in the **<EG_ RUM_ COLLECTOR_ DIR>\tomcat\webapps\rumcollector\WEB-INF\lib** directory (on Windows; on Unix, this will be the **/opt/rum/tomcat/webapps/rumcollector/WEB-INF/lib** directory)
   - Look for the **Client_IP_Header_To_Read**parameter in the file. This parameter points the collector to that request header from which it can glean the client IP address. By default, this parameter is set to *x-forwarded-for*. Based on the type of proxy server / load balancer that is used in your environment for enabling browser-collector communication, you may have to change the value of the parameter.
   - Then, you need to specify the index value of the client IP address that is to be used for geo coding. Typically, when multiple addresses are sent to the collector, they will be included in the request header as a comma-separated list. Each IP address in the list will have an index value, which indicates the position of that IP address in the list. The index value of the first IP address will be 0, the next IP address will be 1, and so on. To enable the collector to pick the IP address to be used for geo coding from the request header, specify the correct index value of that IP address against the **Index_ Position_Of_IP_To_Be_Considered** parameter. By default, this parameter is set to 0, indicating that the first IP address is considered by default and the rest are disregarded by the collector.
   - Finally, save the file.

4. **Can I configure a cluster of eG RUM collectors to ensure high availability?**

   You can configure High Availability for the RUM collector on a Windows HA cluster. This facility is already available for the eG Agent High Availability and is documented in the *eG Installation Guide*. You can mirror the same for RUM collector.

5. **Where should the collector be installed if the web site/web application being monitored is available in a public domain or in a private domain?**
   Please refer to Section **3.2.2**of this document for a detailed response.

6. **How are the performance beacons transmitted to the eG RUM collector?**

   Via HTTP/HTTPS

7. **How frequently are the beacons transmitted to the eG RUM collector?**

   Every time a user accesses a page (where the JavaScript code snippet has been injected) on the web site, the user browser runs the script on the page, computes the load time of the page and where the page spent time when loading, and sends the statistics to the eG RUM collector.

8. **How does the eG RUM collector store the performance metrics it receives?**

The eG RUM collector stores the performance metrics it receives in flat files.

9. **Can I monitor more than one website using one RUM collector?**

   Yes. A single collector can monitor multiple web sites. When adding a *Real User Monitor* component using the eG admin interface, you can assign a collector to that component. By assigning the same collector to multiple *Real User Monitor* components, you can manage multiple web sites/web applications using a single collector.

10. **Does the collector store all the beacons it receives, regardless of the number of web sites it manages, in a single flat file?**

    No. The collector creates a flat file for every web site it manages, and stores beacons pertaining to a web site in the corresponding flat file.

11. **Can a single web site be managed by a cluster of collectors?**

    Currently, we do not support cluster collectors. This means that a single web site can be managed only by a single collector.

12. **How much data can the eG RUM collector store in the flat files?**

    By default, for every web site/web application it monitors, the RUM collector can hold a maximum of *30000* lines of data per measurement period, in the flat files. This limit is however, colnfigurable. To configure the limit, do the following:

    - Edit the **rum.propeties** file in the **<EG_ RUM_ DATA_ COLLECTOR_ INSTALL_ DIR>\tomcat\webapps\rumcollector\WEB-INF\lib** directory (on Windows; on Unix, this will be the **/opt/rum/tomcat/webapps/rumcollector/WEB-INF/lib** directory)
    - Search for the **maxAllowedLinesPerSite**parameter in the file. By default, this parameter is set to *30000*.
    - You can increase or decrease the value of this parameter, depending upon how heavy the user traffic to your web site/web application is and whether/not your RUM collector is well-sized.
    - If you do make changes, save the file.

13. **How often are the flat files cleaned up?**

    As and when the agent requests the collector for measures, the data will be sent to agent and it will be cleaned up immediately from the flat files. If for some reason, the agent is unable to communicate with the collector, the collector will continue to store the metrics received from the browsers in the flat files, until the **maxAllowedLinesPerSite** configuration is reached. Beyond this point, the collector will discard the metrics, so as to ensure the flat file does not grow uncontrollably.

14. **How does the eG RUM collector send the metrics it receives to the eG agent?**

    Whenever the eG agent polls the eG RUM collector for metrics, the collector sends the metrics to the eG agent via HTTP/HTTPS.

15. **What happens to the performance beacons from the browsers, if the eG RUM collector**

is down/unavailable?

In this case, the browsers will discard the beacons.

16. **My RUM collector is running, but the system hosting the collector crashed. Will this affect the load time of the pages where the eG RUM code snippet has been inserted? If so, what do I do to prevent this?**

    Yes. Unavailability of the collector host will indeed impact the load time of the web pages where the code snippet has been inserted.

    The JavaScript code snippet inserted into your web pages carries the location of the **egrum.js** that the browser should run in order to collect load time metrics. Typically, this **egrum.js** script will reside on the RUM collector host. Therefore, at run time, the code snippet will attempt to connect to the RUM collector host to find the **egrum.js** script and run it. In the event that the collector host is unavailable, the code snippet will keep trying to access the host, thus delaying the loading of the corresponding web page.

    To avoid this, before inserting the code snippet into your web pages, make sure you copy the **egrum.js** from the collector host to the local server - i.e., the server that hosts the web site/web application being monitored. Then, update the code snippet with the new location of the **egrum.js**. Finally, insert the code snippet in the web pages to be monitored. This will ensure that the collector host's unavailability does not impact page load time.

17. **How does the collector work in eG manager cluster setup?**

    The collector has no relation to the eG Redundant Manager Cluster setup.

18. **What is the resource requirement of the eG agent and the collector with respect to real user monitoring?**

    Resource requirements will differ from one environment to another, depending upon the traffic to the web site/web application being monitored, frequency of monitoring, and data retention rules.

    To assess the exact resource requirements for your environment, please use the *RUM Sizing Calculator.xls* that is provided to you.

19. **Should the system hosting the eG RUM collector have Internet connectivity to perform Geo location translation?**

    No, Internet connectivity is not required for Geo location translation.

20. **In my environment, a proxy server routes the performance beacons from the browsers to the eG RUM collector. Due to the proxy server configuration, multiple IP addresses are sometimes reported to the collector. How does the collector perform geo location translation in such a case?**

    In this case, you need to configure the eG RUM collector to use only one of the IP addresses for geo coding and drop the rest. For this purpose, follow the steps below:

- Edit the **rum.properties** file in the **<EG_ RUM_ DATA_ COLLECTOR_ INSTALL_ DIR>\tomcat\webapps\rumcollector\WEB-INF\lib** directory (on Windows; on Unix, this will be the **/opt/rum/tomcat/webapps/rumcollector/WEB-INF/lib** directory)
- Look for the **Client_IP_Header_To_Read**parameter in the file. This parameter points the collector to that request header from which it can glean the client IP address. By default, this parameter is set to *x-forwarded-for*. Based on the type of proxy server / load balancer that is used in your environment for enabling browser-collector communication, you may have to change the value of the parameter.
- Then, you need to specify the index value of the client IP address that is to be used for geo coding. Typically, when multiple addresses are sent to the collector, they will be included in the request header as a comma-separated list. Each IP address in the list will have an index value, which indicates the position of that IP address in the list. The index value of the first IP address will be 0, the next IP address will be 1, and so on. To enable the collector to pick the IP address to be used for geo coding from the request header, specify the correct index value of that IP address against the **Index_Position_Of_IP_To_Be_Considered** parameter. By default, this parameter is set to 0, indicating that the first IP address is considered by default and the rest are disregarded by the collector.
- Finally, save the file.

21. **Can a single web site/web application be managed by multiple eG RUM collectors?**

    No, this is not possible.

22. **Can I manage my Intranet and Internet website using the same eG RUM collector?**

    Yes, provided the collector has been configured to receive beacons from the browsers of both the Internet and Intranet users.

6.4 The eG Agent - RUM Collector Communication

1. **How frequently does the eG agent poll the eG RUM collector?**

   The frequency of the tests run by the eG agent determines the frequency of the polling. By default, this is 5 minutes.

2. **Does the eG agent pull all available metrics from the RUM collector at one shot?**

   No. This depends upon how many lines of information a single request from the eG agent is allowed to read from the flat files. By default, the eG agent can read a maximum of *1000* lines simultaneously. This is governed by the **readLines** configuration in the **rum.properties** file, which is set to *1000* by default. If there are more than *1000* lines of data in the flat files, the eG agent then issues multiple requests to the data collector to read the data. You can change the **readLines** configuration, so that the eG agent can read more lines or less lines at one shot.

3. **What happens if the eG agent is down or is unable to connect to the eG RUM collector?**

In this case, the user experience metrics will not be forwarded to the eG agent. The collector however will continue to store the metrics received from the browsers in the flat files, until the **maxAllowedLinesPerSite**configuration is reached. Beyond this point, the collector will discard the new metrics that are collected. Once the agent-collector communication is restored, the collector will send all the metrics that are available in the flat files at that point of time to the agent.

4. **What happens if the eG agent-collector connection is lost when data transmission is in progress?**

   In this case, the RUM collector will clean up all the data that has been collected but not sent to the eG agent yet, from the flat files.

6.5 Configuring eG RUM

1. **When the site traffic is heavy, can I use two agents to collect the data pertaining to a single site?**

   Presently, a single web site can be mapped to only one collector and one agent. Multiple agent/collector option is not available now.

2. **Can the eG agent monitor the user experience with a web site/application that overlays 4 different web/web application servers? If so, how can this be configured?**

   Yes. The eG agent can perform real user monitoring for a web site/web application that is supported by 4 different web/web application servers. To configure this monitoring, do the following:

   - Install and configure a single RUM collector for the target web site/web application.
   - Manage the target as a Real User Monitor component in eG Enterprise.
   - Inject the JavaScript code snippet in each of the web pages that is served by every one of the web/web application servers delivering the target web site / web application.

3. **I have a website that is served from 5 application servers (say, a cluster of app servers). Do I need to create 5 separate 'Real User Monitor' components in eG to monitor them?**

   No, you need to create only **ONE *Real User Monitor*** component. The same code snippet in your JSP or HTML will get deployed on all applicable application servers. This will ensure that the page transmits the beacon, no matter which application server serves that page.

4. **Can I inject the JavaScript code snippet automatically into the pages?**

   Auto-injection is not supported in this release of eG RUM. Plans are on to provide support for this feature in the future releases.

5. **I have a load balanced environment where there are multiple IIS servers serving the same website. Do I need to insert the same eG RUM script on multiple IIS Servers?**

   Yes, you will need to insert the eG RUM script on all IIS Servers that serve the given website.

6. **I have multiple web applications served by ONE IIS server. How do I configure eG RUM script for specific sites?**

   Considering the below mentioned scenario, It is possible to create rewrite rules for each site (A1, A2, B and C) separately with different eG Rum scripts (see). In order to do this, we should select the specific site and create rules for it.

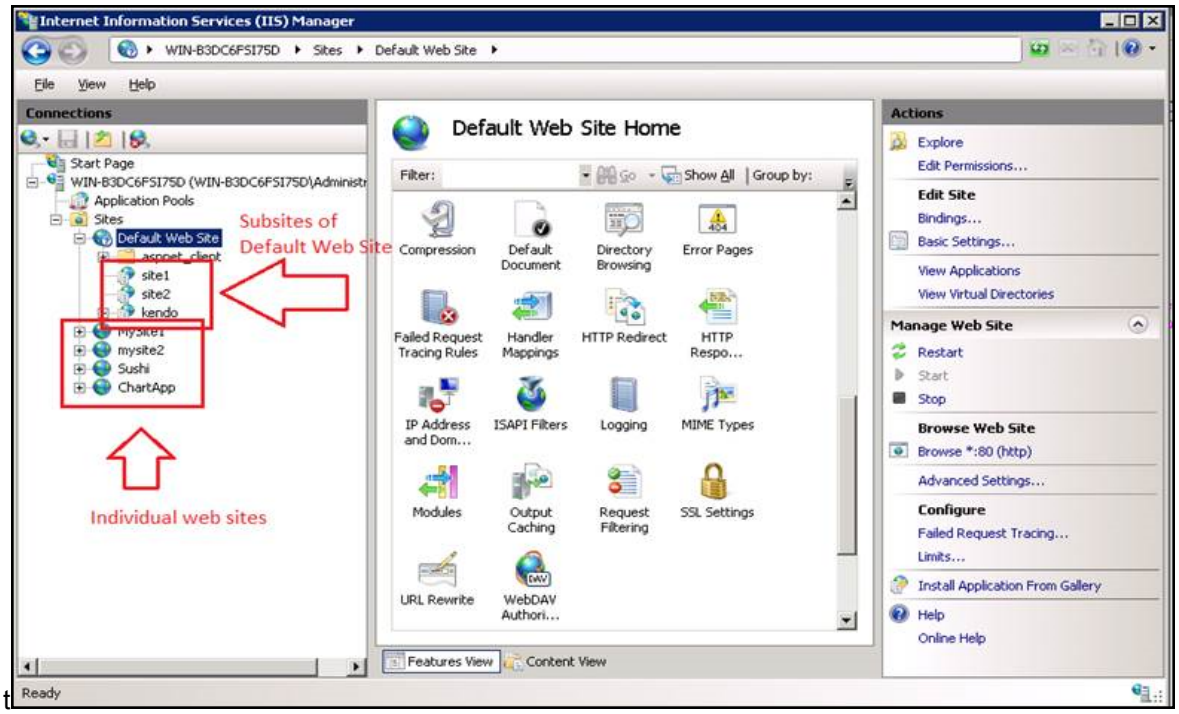
   (A) Default Web Site. (the subsites below shares the same hostname and URL pattern alone differs.)

   (A1) Sub site – site1 (xyz.com/finance/abc.htm)

   (A2) Sub site – site2 (xyz.com/product/abc.htm)

   (B) MySite1 (mysite1.xyz.com)

   (C) mysite2 (mysite2.xyz.com)



7. **Can eG RUM work with reverse proxy?**

   Yes. eG RUM can indeed work with reverse proxy as long as the underlying web server is IIS or Apache. Supported versions of IIS and Apache are available in the Section **3.1**.

8. **Do you have a global button to temporarily turn off or disable eG RUM scripts from all web pages?**

   The answer to this question depends on the underlying technology being instrumented:

   - **Java-based applications:** For Java applications, currently this ability is not supported. We are working on bringing this capability into the product around the first quarter of next year.

- **Dot Net applications served via IIS 7 and above:** Our current product supports enabling RUM script injection via "URL Rewrite" for IIS 7 and above. We have an option to disable the RUM Script injection for all pages by simply disabling the Rules created in the URL Rewrite.
- **SharePoint applications:** For SharePoint applications, currently we add the RUM script in a single page - example : "seattle.master or v4.master". Removing the RUM script from this file will result in disabling RUM for all the web pages.
- **Peoplesoft applications:** For PeopleSoft applications, currently we add the RUM script in a single page - i.e., "PT_COMMON". Removing the RUM script from this file will result in disabling RUM for all the web pages.

9. **How do I inject the eG RUM script in dynamically generated web pages?**

   In the case of a web site / web application that is **not** hosted on IIS, Apache, or any variants, but has common header/footer sections and dynamically generates HTML pages, you can add the code to the Header page of that web site / web application to RUM-enable it. To know how to do this, refer to the Section **3.4** topic.

10. **How do I tell if a page is dynamically generated?**

    Application Admin or the application/website development team only can tell whether the page is dynamically generated or not. This is based on the application/website architecture. It cannot be identified by seeing the browser.

11. **Will IIS injection / rewriting rules be applicable for the dynamically generated website?**

    Yes. IIS Rewrite rule based injection will be able to inject script for dynamically generated pages.

6.6 Monitoring Real User Experience Using eG RUM

1. **I noticed that Google maps are used to represent user locations in the eG RUM dashboard. Does this mean then that the system from which I am accessing the dashboard should have internet connectivity?**

   Yes. Because, without Internet connectivity, you will not be able to view the user location maps in the eG RUM dashboard.

2. **I have hosted my business online, and have partnered with many other brands to host ads related to my business on their web site. This way, any visitor to their web site can click on an ad and switch to my site. I would like to know which of these partnerships has been most beneficial to me? In other words, I want to know the web sites that have contributed to the maximum number of hits to my web site. Can I glean this information from eG RUM?**

   The detailed diagnostics provided by eG Enterprise reveal which pages in your web site were viewed and the URL from which each page view was launched. The latter is called the 'Referer URL'. In the case of your scenario, this will be the URL of your partners. While eG provides this information at a per-page view level, it presently does not offer you any means to figure out the count of page views per

Referer URL. This means that using eG, you will not be able to tell which referer/partner was responsible for the maximum hits to your web site.

3. **Will eG alert me when there are no hits or more hits to a web site than expected?**

   Yes. eG embeds efficient self-learning capabilities that enable such intelligent alerting. By tracking the page views to your web site over a period of time, eG automatically computes the upper and lower bounds of traffic to the web site – in other words, eG automatically determines how high and how low the count of page views typically go. When any of these auto-computed baselines are violated, eG alerts users. So, if you have a web site that normally receives a moderate number of page requests, but suddenly receives none, then eG's auto-computed lower bound will be violated, resulting in an alert. Likewise, if that web site receives an unusually high number of page requests, eG's auto-computed upper bound will be violated, resulting in an alert. You also have the option of manually setting static threshold values, in the place of the auto-computed ones.

4. **Can eG RUM highlight the number of unintended ads loaded on the web page, which may be adversely impacting user experience?**

   No. This is currently not supported.

5. **How does eG RUM handle time zone differences between the various components of its architecture?**

   Typically, the eG RUM collector timestamps the beacons it receives based on the eG manager's time zone. Since the eG agent too, by design, synchronizes its time with the eG manager, the agent does not make any changes to the timestamp of the beacons when it pulls them from the RUM collector and sends them to the eG manager. However, if a different time zone has been set in eG for a user registered with the eG Enterprise system, then, when that users logs into the eG monitoring console, the RUM metrics he/she views in the console will be timestamped according to that user's time zone setting. Historical metrics viewed by this user in the eG Reporter console will also be aligned with that user's time zone setting only.

6. **How does the eG RUM script calculate the page download time and its break up?**

   Navigation Timing is a JavaScript API for accurately measuring performance on the web. The API provides a simple way to get accurate and detailed timing statistics - natively - for page navigation and load events. It's available in all recent browsers.

   The injected eG RUM Script will make use of the Navigation Timing API to gather the load time of a web page and the break up of the load time.

7. **What are the time components that are not included in the page load time reported by eG RUM?**

   The time taken for running the following will not be included in the page load time:
   - Flash
   - Silverlight
   - Applet

- Java FX
- java webstart
- activex
- HTML5 videos
- Flash based videos

This is because, the components above run in their own runtime (typically as a browser plug-in).

8. **Is it true that the eG RUM can tell from which countries users are accessing a web site/web application?**

   Yes, it is true. The eG RUM can automatically discover the countries, cities, and regions from which the users to a web site/web application are coming, and can report the experience of users from every country, city, and region so discovered.

9. **How does eG RUM discover the geographic location of users?**

   This intelligence is embedded in the eG RUM collector. Once the collector receives user experience metrics from the browsers, it scans the metrics and extracts the client IP addresses – i.e., IP addresses of the user browsers/devices - from it. The collector then performs IP address to Geo location translation to discover the location of the users.

   To perform geo location translation, the collector uses the following:

   - A third-party binary database called *MaxMind*
   - A Geo Location Mapper file (which is an XML file)

   Typically, every incoming client IP address is first checked for a match against the *MaxMind* database. If a match is found, then it is deemed to be a public IP address. If a matching IP address is not found in the *MaxMind* database, then the collector checks the private IP address ranges (with subnet details) present in the Geo Location Mapper file for a match. If a match is found, then the IP address is deemed to be a private IP address.

10. **Does the page loading time include the loading time of all resources such as images, videos, etc.?**

    The page loading time typically includes the loading time of resources such as images, JavaScripts, stylesheets (CSS), etc. However, videos are not included in the page loading time. This is because, usually, video playback is asynchronously rendered independent of the page load completion. In other words, the video playback will commence only after the DOM Complete event is triggered. This is why, the page load time does not include the time to render videos.

11. **If user requests are routed through a proxy server to the web server, can eG RUM report the time spent by the requests on the proxy server?**

No. eG RUM cannot give the breakdown as asked above. eG RUM (or any RUM solution for that matter) will report the total server backend time – however, the breakdown of this time will not be visible to RUM since it computes metrics from a JavaScript sitting inside the browser.

12. **Can eG RUM provide additional transaction details such as the city from which the request originated and the telecom provider?**

    eG RUM currently uses Maxmind GEO IP2 City to provide geo mapping. Using this tool, eG RUM captures and reports the cities from which requests to the web site/web application being monitored were received. For each city, eG RUM reports the average load time of the page view requests from that city and where the requests spent time.

    However, eG RUM presently does not report telecom provider details. Note that this will require purchasing a separate database from Maxmind on eG's part (https://www.maxmind.com/en/geoip2-isp-database). Capturing ISP or Telecom provider is not in our roadmap for the immediate future.

13. **Can eG RUM support AJAX web applications? If so, then will the RUM script injected in the web pages of such applications impact Ajax?**

    Yes, RUM can support AJAX web applications. However, note that the underlying application should have a combination of traditional Base Page with additional AJAX requests. The reason is that our current product is geared towards traditional web apps which have a combination of base page with additional AJAX. Think of a site such as Amazon where the product page is a traditional request-response base page with additional ajax to fetch "cross-sell" product on the "You may also like…" section. Therefore, If the customer has a 100% pure AJAX application (such as a Single Page Application e.g. AngularJS or even ExtJS), such request URLs will NOT be shown on the reports. The 'Web Site' test will also not consider these AJAX requests. On the other hand, such requests can be seen on the layer model in the 'Page Type test' and in the RUM dashboard.

    No, the RUM script will NOT impact AJAX functionality either functionally or from an elapsed time performance overhead perspective.

7

Conclusion

This document has described in detail the monitoring paradigm used and the measurement capabilities of the **eG Real User Monitor**. For details of howto 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.