



The eG Real User Monitor

eG Innovations Product Documentation

www.eginnovations.com



Table of Contents

CHAPTER 1: INTRODUCTION	1
1.1 How to Measure End User Experience?	2
1.2 How does eG Enterprise Monitor End-User Experience with Web Sites/Web Applications?	2
CHAPTER 2: REAL USER MONITORING USING EG ENTERPRISE	4
2.1 Qualities of an Ideal Real User Monitor	4
2.2 Challenges Faced when Using Traditional RUM Tools	5
2.3 The eG Approach to Real User Monitoring	6
2.4 What the eG RUM Reveals	7
2.5 The eG RUM Advantage	8
CHAPTER 3: HOW TO USE THE EG REAL USER MONITOR	11
3.1 Pre-requisites for Real User Monitoring using eG	11
3.2 Installing and Configuring an eG RUM Collector	17
3.2.1 Pre-requisites for Installing an eG RUM Collector	17
3.2.2 Where to Install the eG RUM Collector?	18
3.2.3 Installing the eG RUM Collector	22
3.2.4 Starting/Stopping the RUM Collector	35
3.2.5 SSL-enabling the eG RUM Collector	35
3.2.6 Registering the New eG RUM Collector with the eG Manager	63
3.3 Managing the Web Site/Web Application to be Monitored	65
3.4 Instrumenting the Web Site / Web Application to be Monitored	67
3.4.1 Instrumenting a Web Site or Web Application on IIS using URL Rewrite	69
3.4.2 Instrumenting an Apache Web Server Using Container-Assisted Injection	92
3.4.3 Instrumenting an NGINX Server Using Container-Assisted Injection	96
3.4.4 Instrumenting an Oracle HTTP Server Using Container-based Injection	97
3.4.5 Instrumenting a Java-based Web Site / Web Application	99
3.4.6 Instrumenting a DotNet-based Web Site / Web Application	100
3.4.7 Instrumenting SharePoint	101
3.4.8 Instrumenting PeopleSoft	112
3.4.9 Enabling RUM for Atlassian Confluence	114
3.4.10 Instrumenting a Web Site or Web Application Using a NetScaler Rewrite Policy	122
3.4.11 Instrumenting a Web Application Using an F5 BIG-IP Load Balancer	131
3.4.12 Instrumenting SAP Fiori Apps	137
3.4.13 Instrumenting Using the JavaScript Code Snippet	180
3.5 Modifying the Test Configuration	181
3.6 Monitoring using the eG Real User Monitor	182
3.6.1 Web Site Test	184

3.6.2 Browsers Test	216
3.6.3 Devices Test	247
3.6.4 Page Types Test	277
3.6.5 Page Groups Test	308
3.6.6 The Location Tests	338
3.6.7 Detailed Diagnostics	429
3.6.8 The RUM Dashboard	441
CHAPTER 4: TROUBLESHOOTING USING THE RUM LOGGER	459
4.1 Configuring Logging for the eG RUM Collector	459
4.2 Configuring Logging for the eG Agent Communicating with the eG RUM Collector	461
CHAPTER 5: TROUBLESHOOTING FAILURE OF THE EG RUM TESTS	465
CHAPTER 6: THE EG REAL USER MONITOR FAQ	469
6.1 Introduction to eG RUM	469
6.2 eG RUM Licensing	470
6.3 The eG RUM Collector	471
6.4 The eG Agent - RUM Collector Communication	481
6.5 Configuring eG RUM	482
6.6 Monitoring Real User Experience Using eG RUM	485
ABOUT EG INNOVATIONS	495

Table of Figures

Figure 2.1: How the eG RUM works	6
Figure 3.1: Page load events captured by the Navigation Timing API	13
Figure 3.2: Downloading the egrum.js and copying the RUM Header script to a text editor	16
Figure 3.3: 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	19
Figure 3.4: 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	20
Figure 3.5: 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	21
Figure 3.6: 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	22
Figure 3.7: A RUM collector deployment model where the target web application is accessed by intranet users and the eG agent is on the cloud	22
Figure 3.8: The Welcome screen	23
Figure 3.9: The End User License Agreement	24
Figure 3.10: Selecting a JDK version to use for collector installation	25
Figure 3.11: Setup enquiring the availability of JDK in the environment	26
Figure 3.12: Specifying the location of the Java home directory for installing the eG RUM collector	26
Figure 3.13: Selecting the IP address/host name of the collector	27
Figure 3.14: Manually specifying the IP address/host name of the collector host	27
Figure 3.15: Indicating whether/not to SSL-enable the eG RUM collector	28
Figure 3.16: Specifying the install directory of the eG RUM collector	28
Figure 3.17: A summary of the specifications	29
Figure 3.18: Setup program indicating the completion of the eG RUM collector installation	29
Figure 3.19: The WebLogic console	31
Figure 3.20: List of applications deployed on the server	32
Figure 3.21: Clicking the 'upload your file(s)' link	32
Figure 3.22: Choosing the rumcollector.war file to be uploaded	33
Figure 3.23: Message confirming the successful file upload	33
Figure 3.24: Clicking the Next button to proceed	34
Figure 3.25: Clicking Finish to end the war file installation	34
Figure 3.26: Successful installation of the eG RUM Collector	35
Figure 3.27: Requesting for a certificate	40
Figure 3.28: Downloading the certificate	41
Figure 3.29: The "Certificate error" that the browser reports	56
Figure 3.30: Selecting the option to install the certificate on the browser host	57
Figure 3.31: Welcome screen of the Certificate Import Wizard	58
Figure 3.32: Choosing to place the certificate in a specific store	59
Figure 3.33: Storing the certificate in the Trusted Root Certificate Authorities store	60

Figure 3.34: The chosen store displayed	61
Figure 3.35: Finishing the import	62
Figure 3.36: A warning message that appears when importing a certificate issued by an internal CA	62
Figure 3.37: A message box informing you that the certificate has been successfully imported	63
Figure 3.38: The login screen of the eG manager, without the 'Certificate error'	63
Figure 3.39: List of RUM collectors that pre-exist	64
Figure 3.40: Adding a new RUM collector	64
Figure 3.41: Choosing to add a Real User Monitor	65
Figure 3.42: Adding a Real User Monitor	66
Figure 3.43: Viewing the code snippet to be injected into the monitored web pages	67
Figure 3.44: The options available for RUM-enabling a web site / web application and when those options become applicable	67
Figure 3.45: Accessing the Compression menu	70
Figure 3.46: Checking whether static compression is enabled	71
Figure 3.47: Setting the dynamicCompressionBeforeCache property to false	72
Figure 3.48: Accessing the URL Rewrite option	73
Figure 3.49: Choosing to add a new rule	73
Figure 3.50: Selecting the Blank Rule option	74
Figure 3.51: Defining the properties of the new precondition	75
Figure 3.52: Providing the condition input, input string, and pattern	75
Figure 3.53: Entering the pattern for the rule	76
Figure 3.54: Specifying the line of code to be auto-injected in the head	77
Figure 3.55: Applying the changes	77
Figure 3.56: Accessing the URL rewrite option	78
Figure 3.57: Choosing to add a new rule	78
Figure 3.58: Selecting the Blank Rule option	79
Figure 3.59: Adding the eG RUM Body Rule	80
Figure 3.60: Specifying the line of code to be auto-injected in the body	81
Figure 3.61: Applying the changes	81
Figure 3.62: Auto-injecting the code snippet in web pages that match specific URL patterns	82
Figure 3.63: Verifying whether a web application is a 32-bit or a 64-bit application	83
Figure 3.64: Installing the tracing role service on Windows 2012	84
Figure 3.65: Installing the tracing role service on Windows 8 or 8.1	85
Figure 3.66: Installing the tracing role service on Windows 2008/2008 R2	86
Figure 3.67: Installing the tracing role service on Windows 7 or Vista	87
Figure 3.68: Clicking Failed Request Tracing	88
Figure 3.69: Enabling tracing	88
Figure 3.70: Clicking Failed Request Tracing Rules	89
Figure 3.71: Selecting the content type to trace	90

Figure 3.72: Defining trace conditions	91
Figure 3.73: Selecting one/more trace providers	91
Figure 3.74: Selecting one/more verbosity levels	92
Figure 3.75: Editing the nginx.conf file	97
Figure 3.76: Adding the Sharepoint site as a Real User Monitor component	102
Figure 3.77: The Home page of the Sharepoint site to be monitored	103
Figure 3.78: Selecting the Site settings option from the Home page	104
Figure 3.79: Selecting the Master pages option	104
Figure 3.80: Locating the seattle.master file on Sharepoint 2013	105
Figure 3.81: Checking out the master file	105
Figure 3.82: Downloading a copy of the master file	106
Figure 3.83: Inserting the Javascript code in the master file	107
Figure 3.84: Uploading the modified master file	108
Figure 3.85: Choosing the modified master file for uploading	108
Figure 3.86: Saving the changes to the master file	109
Figure 3.87: Confirming that the timestamp of the master file reflects the time of modification	109
Figure 3.88: Choosing to manage web applications	110
Figure 3.89: Selecting the Sharepoint web application to be monitored	110
Figure 3.90: Selecting the 'General Settings' option	111
Figure 3.91: Setting the 'Application –Layouts ...' flag to Yes	111
Figure 3.92: Logging into PeopleTools	112
Figure 3.93: Searching for an HTML definition	113
Figure 3.94: Inserting the JavaScript code snippet in the HTML definition	114
Figure 3.95: The RUM script that is to be injected into the HTTP responses	115
Figure 3.96: Selecting the General Configuration option	116
Figure 3.97: Clicking on the Cache Management option	116
Figure 3.98: Clicking on the Show advanced view link	116
Figure 3.99: Flushing all page-related caches	117
Figure 3.100: Downloading the egrum.jar file	118
Figure 3.101: Selecting the Add-ons option	119
Figure 3.102: Clicking on the Upload add-on link	119
Figure 3.103: Uploading the add-on	119
Figure 3.104: A message box confirming the successful upload	120
Figure 3.105: Selecting the General Configuration option	120
Figure 3.106: Clicking on the Cache Management option	121
Figure 3.107: Clicking on the Show advanced view link	121
Figure 3.108: Flushing all page-related caches	121
Figure 3.109: The RUM script that is to be injected into the HTTP responses	122

Figure 3.110: Selecting the Configure Basic Features option	124
Figure 3.111: The Configure Basic Features page	124
Figure 3.112: Clicking on the Add button to add a new rewrite action	125
Figure 3.113: Adding a rewrite action to replace the body of the HTTP response	125
Figure 3.114: Replacing the body of the HTTP response with the RUM code snippet (applicable to NetScaler appliances before v12)	126
Figure 3.115: Replacing the body of the HTTP response with the RUM code snippet (applicable to NetScaler appliances of v12 and above)	127
Figure 3.116: Clicking the Add button to add a new rewrite policy	128
Figure 3.117: Creating a new rewrite policy	128
Figure 3.118: Binding the rewrite policy	129
Figure 3.119: Clicking on the Add Binding button	130
Figure 3.120: Adding a Binding	130
Figure 3.121: Completing the rewrite policy rule creation	131
Figure 3.122: The RUM script that is to be injected into the HTTP responses	132
Figure 3.123: Selecting the HTML -> Rules option in the F5 console	132
Figure 3.124: Selecting the Prepend HTML option	133
Figure 3.125: Specifying the name of the Prepend HTML rule	133
Figure 3.126: Specifying a Match Tag Name	134
Figure 3.127: Copying the RUM code snippet to the HTML to Prepend text area	134
Figure 3.128: Clicking on the Create New Profile button	135
Figure 3.129: Creating a new HTML profile	135
Figure 3.130: Associating the HTML Rule with the HTML Profile	135
Figure 3.131: List of virtual servers	136
Figure 3.132: Selecting the HTML profile to be enabled on the Virtual server	136
Figure 3.133: Error that appears when attempting to update the virtual server details	137
Figure 3.134: Disabling Strict Updates for an Application Service	137
Figure 3.135: Managing a SAP Fiori APP using the eG admin interface	138
Figure 3.136: Logging into the SAP Cloud Platform	139
Figure 3.137: Clicking on SAP Web IDE Full-Stack	140
Figure 3.138: Enabling the SAP Web IDE Full-Stack service	140
Figure 3.139: Enabling SAP Fiori Launchpad Extensibility	141
Figure 3.140: Choosing to create a new Project from a template	142
Figure 3.141: Choosing to create a new SAP Fiori Launchpad Plugin	143
Figure 3.142: Providing the name of the new project	144
Figure 3.143: Assigning an ID and title to the new plugin	144
Figure 3.144: Creating a lib folder under the eGRUMPlugin folder	145
Figure 3.145: Creating the egrum.js file under the lib folder	146

Figure 3.146: Copying the contents of the downloaded egrum.js file to the empty egrum.js file in the SAP console	147
Figure 3.147: Editing the manifest.json file	148
Figure 3.148: Initiating the deployment of the eGRUMPlugin to the SAP Cloud platform	148
Figure 3.149: Deploying the eGRUMPlugin	149
Figure 3.150: A message confirming the successful deployment of the plugin	149
Figure 3.151: Clicking the Portal service link in the Cockpit	150
Figure 3.152: Clicking on the Go to Service link	150
Figure 3.153: Choosing the site to be RUM-enabled	151
Figure 3.154: Clicking the '+' icon to add the eGRUMPlugin to the chosen site	152
Figure 3.155: Clicking the App Resource Browse button to look for the eGRUMPlugin	152
Figure 3.156: Selecting the eGRUMPlugin from the list of App Resources	153
Figure 3.157: Setting Shell Plugin as the App Type	154
Figure 3.158: Clicking on the Catalogs icon	155
Figure 3.159: Clicking on the '+' icon to add a catalog	155
Figure 3.160: Selecting the catalog in which the eG RUM Plugin is to be added	156
Figure 3.161: The selected catalog displayed under Assigned Catalogs	157
Figure 3.162: Publishing the site	157
Figure 3.163: Verifying successful loading of egrum.js and the transmission of beacons to collector using Developer tools	158
Figure 3.164: Logging into the SAP Cloud Connector	159
Figure 3.165: Clicking on the option that corresponds to the SAP Cloud Platform subscription in use	159
Figure 3.166: Creating a new subaccount	160
Figure 3.167: Determining the Subaccount name	161
Figure 3.168: Clicking the Cloud to On-Premise option	161
Figure 3.169: The new mapping appended to the Mappint Virtual To Internal System list	162
Figure 3.170: Selecting the new mapping and clicking the '+' icon to add a new resource	162
Figure 3.171: Adding a new resource	163
Figure 3.172: Checking connectivity	163
Figure 3.173: Selecting the Destinations option from the Connectivity node	164
Figure 3.174: Clicking the New Destination button	164
Figure 3.175: Configuring a new destination	165
Figure 3.176: Checking connectivity to the new destination	165
Figure 3.177: Launching SAP Web IDE	166
Figure 3.178: Enabling SAP Fiori Launchpad Extensibility	166
Figure 3.179: Selecting the option to create a new project from template	167
Figure 3.180: Selecting the SAP Fiori Launchpad Plugin	167
Figure 3.181: Specifying the new project name	168
Figure 3.182: Providing the plugin ID and title	168

Figure 3.183: Creating a sub-folder under eGRUMPlugin	169
Figure 3.184: Naming the sub-folder 'lib'	169
Figure 3.185: Creating a file under the lib folder	170
Figure 3.186: Specifying the new file name as egrum.js	170
Figure 3.187: Pasting the contents of the downloaded egrum.js file into the empty egrum.js file	171
Figure 3.188: Editing the manifest.json file when RUM-enabling SAP Fiori Apps On-Premise	172
Figure 3.189: Deploying the eGRUMPlugin to the SAPUI5 ABAP Repository	172
Figure 3.190: Selecting the deployment destination	173
Figure 3.191: Deploying a new application	173
Figure 3.192: Finishing the application deployment	174
Figure 3.193: Activating the service	174
Figure 3.194: Clicking on the + icon to create a new catalog	175
Figure 3.195: Creating a new catalog	176
Figure 3.196: Clicking on the Target Mapping option	176
Figure 3.197: Clicking on the Create Target Mapping option	177
Figure 3.198: Creating a new target mapping	178
Figure 3.199: Creating a new role	178
Figure 3.200: Assigning the newly created catalog to the new role	179
Figure 3.201: Verifying whether the eG RUM configuration for SAP Fiori Apps for On-Premise setup is working or not	180
Figure 3.202: Inserting the code snippet in a web page	181
Figure 3.203: Layer model of the eG RUM component	182
Figure 3.204: Tests mapped to the Real User Monitoring layer	183
Figure 3.205: The page loading process	184
Figure 3.206: Sample detailed diagnosis of a measure reported by the Web site test	429
Figure 3.207: The detailed diagnosis of the Page views measure	431
Figure 3.208: The detailed diagnosis of the Average page load time measure	431
Figure 3.209: The detailed diagnosis of the JavaScript errors measure	432
Figure 3.210: The detailed diagnosis of the Slow page views measure	432
Figure 3.211: The detailed diagnosis of the Tolerating page views measure	432
Figure 3.212: The detailed diagnosis of the Frustrated page views measure	433
Figure 3.213: The detailed diagnosis of the Satisfied page views measure	433
Figure 3.214: The detailed diagnosis of the Desktop page views measure	434
Figure 3.215: The detailed diagnosis of the Mobile page views measure	434
Figure 3.216: The detailed diagnosis of the Tablet page views measure	435
Figure 3.217: The RUM Transaction Details page displaying the entire flow of the page request/transaction	436
Figure 3.218: The Resource Details tab page revealing the resources that took too long to be downloaded from the server	436
Figure 3.219: Resource fetch time break-up	437

Figure 3.220: The transaction flow indicating the reason why Network connection time is high	438
Figure 3.221: The transaction flow chart displaying the break-up of Content Download Time	438
Figure 3.222: The transaction flow displaying page information	439
Figure 3.223: The transaction flow displaying user information	440
Figure 3.224: The RUM Transaction Details for an Ajax call	440
Figure 3.225: Errors displayed in the Error Details tab page	441
Figure 3.226: The Real User Monitor Dashboard	442
Figure 3.227: Details of slow page views where the URLs of the pages are provided along with the breakup of each page	443
Figure 3.228: The RUM dashboard of a chosen web site/application	444
Figure 3.229: The array of icons available in the left panel of the application-specific RUM dashboard	444
Figure 3.230: The RUM dashboard revealing the user experience per country	445
Figure 3.231: User experience analysis across geographies	445
Figure 3.232: Customizing the Geo map	446
Figure 3.233: Viewing the user experience statistics of a particular country in the geo map	447
Figure 3.234: Zooming into the regions within a country	447
Figure 3.235: Zooming into the cities within a region	448
Figure 3.236: Response Time Breakup of users from a chosen geography using a chosen device type	448
Figure 3.237: Slowest page groups in a chosen geography	449
Figure 3.238: Response time analysis	450
Figure 3.239: Response time breakup for a chosen URL across device types	450
Figure 3.240: Break-up of Network Connection Time and the top-5 URLs impacted by high network connection time	451
Figure 3.241: Break-up of Content Download Time and the top-5 URLs impacted by high content download time	452
Figure 3.242: Response time breakup analysis by page group	452
Figure 3.243: User experience analysis by channel	453
Figure 3.244: Analysis across URLs	454
Figure 3.245: Summary of slow page views	454
Figure 3.246: Analysis of a chosen URL	455
Figure 3.247: Usage analysis	455
Figure 3.248: Trend and distribution analysis	456
Figure 3.249: Error analysis	457
Figure 3.250: Identifying error-prone browsers and URLs, and correlating error views with total page views	458
Figure 5.1: Developer Tools console revealing whether/not the JavaScript is successfully inserted in a web page	465
Figure 5.2: Determining whether/not the browser is sending performance beacons to the RUM collector	466
Figure 6.1: Two web sites sending beacons to the RUM collector using a single Site ID	470
Figure 6.2: Two web sites sending beacons to the RUM collector using different Site IDs	471

Figure 6.3: Selecting the Add Rule option	474
Figure 6.4: Selecting Reverse Proxy as the inbound rule template	474
Figure 6.5: Specifying the server name / IP address of the RUM collector	475
Figure 6.6: Editing the new inbound rule	475
Figure 6.7: Specifying the URI pattern to capture and the Rewrite URL it should be redirected to	476

Chapter 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 'simulated' 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?

Chapter 1: Introduction

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

Chapter 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.

- Section 2.1
- Section 2.2
- Section 2.3
- Section 2.4
- Section 2.5

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.

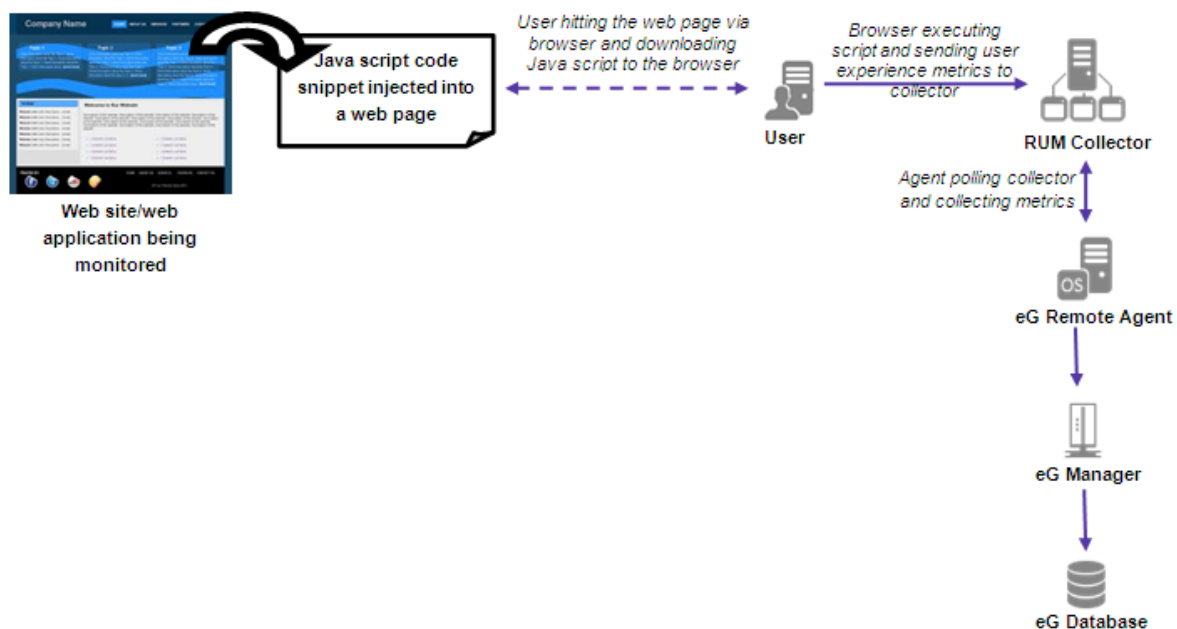


Figure 2.1: How the eG RUM works

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 complains 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!

Chapter 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 and fully/partially supporting the following web technologies/frameworks:

Fully supported web technologies/frameworks	Partially supported web technologies/frameworks ¹
Java, .NET, PHP, NodeJS, Python, Ruby on Rails, HTML, XHTML, SHTML, Iframe, Ajax, Object, Frameset, Embed, Angular 1.x, 2 and above	Backbone, Ember, React, jQuery, KnockoutJS, Dojo, CanJS, Polymer, Mithril, Ambersand, Flight, Vue.js, Marionette.js, TroopJS+RequireJS, Aurelia, Spine, Matreshka.js, Redux, Rxjs

1 - These are Single Page Apps (SPA). If the target web application supports these technologies, then eG RUM will be able to report the page load time, resource load time, and Ajax metrics for that target. However, virtual page metrics will not be reported for the same.

- A **Javascript-enabled** browser that ideally supports both the **Navigation Timing API** and the **Resource Timing API**. The **Navigation Timing API** provides data that can be used to measure the performance of a web site. The **Resource Timing API** provides a way to retrieve and

analyze detailed network timing data regarding the loading of an application's resource(s) (eg., image, script, XMLHttpRequest, <SVG> etc.).

The table below lists the popular browsers, the browser versions that eG RUM supports fully and partially, and the versions that are not supported.

Browser	Versions Fully Supported ¹	Versions Partially Supported ²	Versions Not Supported ³
Internet Explorer*	10, 11	6, 7, 8, 9	5 and below
Microsoft Edge	12, 13, 14, 15, 16		1
Chrome	24 and above	6 to 23	5 and below
Mozilla Firefox	35 and above	7 to 34	6 and below
Safari	11 and above	8 to 10	7 and below
Opera	15 and above		14 and below
IOS Safari	8, 8.1, 9.2 and above		7 and below, and between 8.2 and 9.2
Android browser	4.4 and above	4 and 4.3	3 and below
Blackberry Browser		10	
Opera Mobile	37		
Chrome for Android	59		
Firefox for Android	54		
IE Mobile	10 and above		
UC browser for Android	11.4		
Samsung Internet	4 and above		
QQ Browser	1.2		
Baidu Browser	7.12		

1 - These versions of browsers support both the Navigation Timing API and Resource Timing API

2 - These versions of browsers support only the Navigation Timing API and not the Resource Timing API. Because of this, eG RUM will not be able to report on the load time of individual resources.

3 - These versions do not support both the Navigation and Resource Timing APIs

* - Note the following with respect to Internet Explorer:

- 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.
- The Navigation Timing API typically exposes several properties that offer information about the time at which different page load events happen (see Figure 3.1) - eg., the `requestStart` event, the `responseStart` event, etc.

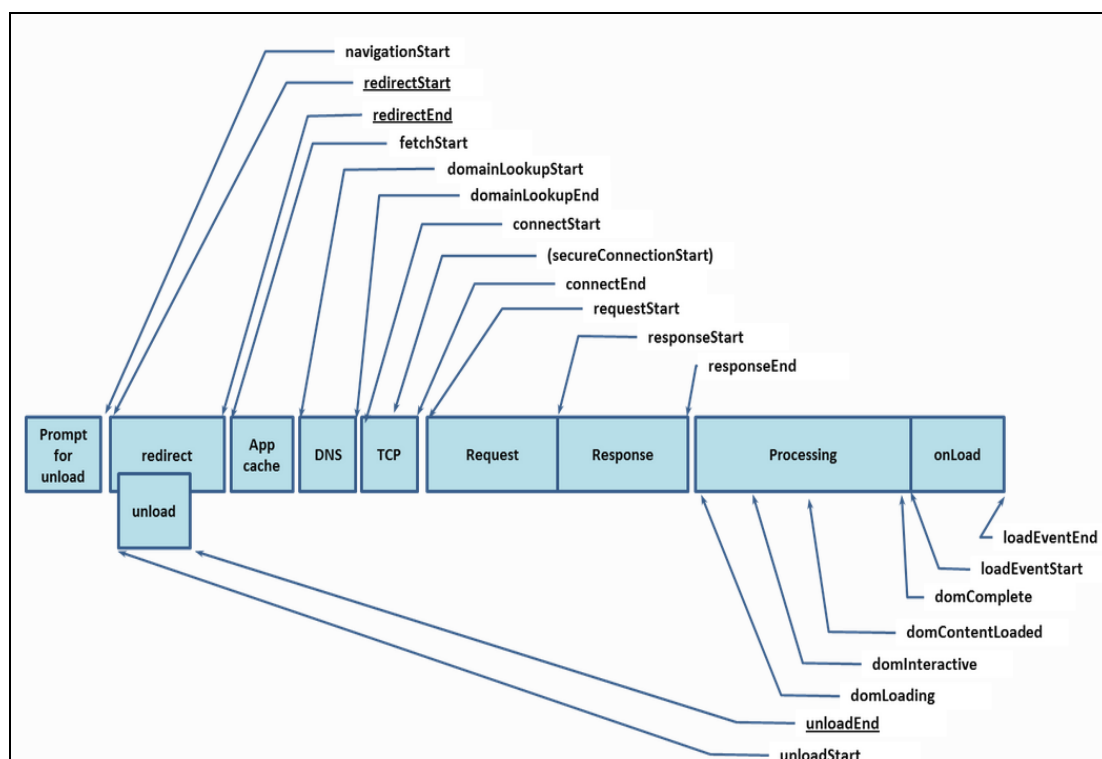


Figure 3.1: Page load events captured by the Navigation Timing API

eGRUM uses the time stamps provided by the Navigation Timing API to compute and report the duration of page load events, so you can accurately identify where page loading is bottlenecked.

The Navigation Timing API on Internet Explorer (IE) v11 in particular reports an incorrect time stamp for the `requestStart` event of the page loading process (see Figure 3.1). As a result, for page view requests initiated from IE 11 browsers alone, eGRUM will report incorrect values for the *Average network time* and *Average server time* measures.

This issue was noticed in IE 11 in April 2019. It is recommended that you track hot fixes/patches released by Microsoft post April 2019, study the release notes of such fixes/patches, and determine if this bug has been fixed in any. If so, then you are advised to apply that fix/patch on IE 11 to resolve the issue.

Until then, we recommend that you use the following workaround to accurately measure the *Average server time* of a page view request.

- Deploy eG BTM (Java/.NET, as the case may be) on the backend server hosting the target web site/web application.
- Use eG BTM to trace the path of transactions to the target web site/web application and enable the **(Java or .NET) Business Transactions** test to capture metrics on transaction performance.
- Next, use the detailed diagnostics reported by eG RUM to identify the page view requests coming from the IE 11 browser. Make a note of the values in the **Request time**, **URL** and **Query Params** columns of detailed diagnosis.
- Then, search the detailed diagnostics of the **(Java or .NET) Business Transactions** test for transactions with the same **URL**, **Request time**, and **Query Params** as reported by eG RUM.
- The response time that eG BTM reports for each of these transactions is the *Average server time* of those transactions.

Note that this workaround applies only for those transaction URLs that are captured and reported as part of detailed diagnostics.

- User traffic should come from the following devices only:
 - PC
 - Server
 - Laptop
 - Mobile
 - Tablet
- 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 *eG Manager and Database Sizing Calculator*.

- 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 *eG Manager and Database Sizing Calculator*.

- By default, the eG RUM script (egrum.js) is bundled with the eG RUM Collector only. Because the RUM script is co-located with the collector, an unexpected collector outage can impact the performance and load time of web pages injected with the RUM script. To ensure fast loading and high availability of such web pages, and to assure users of a superior experience, we strongly **recommend that you place the egrum.js file in a highly available location**. The steps to achieve this are as follows:
 - First, **pick a highly available location** in your environment. This can be any of the following:
 - A web server;
 - A web application server;
 - A load balancer;
 - The public cloud
 - Next, you need to download the **egrum.js** script file. For that, first login to the eG admin interface.

If a *Real User Monitor* component is already managed in the eG Enterprise system, then, follow the Infrastructure -> Components -> Add/Modify menu sequence, pick *Real User Monitor* as the **Component type**, and click the **Modify** button corresponding to the RUM component for which the *egrum.js* is to be relocated. In the figure that appears next (see Figure 3.2), click the **Download** button against *egrum.js* to download it. Also, copy the RUM header script displayed in the **Include this line in your . . .** text area to Notepad or any text editor.

To instrument your html pages use the eG javascript agent

Download this if you have a Java web application that emits HTML to the front-end. Examples: JSF, JSP, Groovy:	egrum.jar	<button>Download</button>
Download this if you have a .NET web application such as ASPNET MVC or WebForms:	Eg.RUM.dll	<button>Download</button>
Download this if you can edit the source code of your web pages. Examples: Plain HTML, JSP with embedded HTML, PHP with embedded HTML:	egrum.js	<button>Download</button>
Download this if want to embed this egrum.js into packaged applications such as SAP Fiori:	Code snippet + egrum.js	<button>Download</button>

Select the source file type on which eG RUM JavaScript code snippet to be included

Include this line in your pages (html/jsp/aspx) immediately after the head tag

☒ HTML
 ☐ Javascript

```
<!-- RUM Header -->
<script charset="UTF-8" type="text/javascript"> window['egrum-start_time'] = new
Date().getTime();
window['Site_Name'] = '37951ab5-af89-499b-85c5-b6bd240a153c-1575469485811';
window['beacon-url'] = 'https://192.168.8.26:7077';
</script>
<script src="https://192.168.8.26:7077/rumcollector/egrum.js" async> </script>
```

Figure 3.2: Downloading the *egrum.js* and copying the RUM Header script to a text editor

If a *Real User Monitor* component is yet to be managed, then, follow the Infrastructure -> Components -> Add/Modify menu sequence, pick *Real User Monitor* as the **Component type**, and click the **Add New Component** button to add the component. In the figure that appears next, provide a nick name for the RUM component, assign a collector and remote agent to it, and click **Add** to add the component. Figure 3.2 will then appear. Here, click the **Download** button against *egrum.js* to download it. Also, copy the RUM header script displayed in the **Include this line in your . . .** text area to Notepad or any text editor. Finally, click the **Add** button to add the component.

- Once the *egrum.js* file is downloaded successfully, **copy it to any folder in a highly available location**.
- Next, switch to the editor to which the RUM header script has been copied. Change the location of the *egrum.js* in the header to reflect its new location. For instance, in the sample header script below, you need to change the text highlighted in Red to indicate the new location.


```
<!-- RUM Header -->

<script charset='UTF-8' type='text/javascript'> window['egrum-start_time'] = new
Date().getTime();

window['Site_Name'] = 'fbe3ef39-c0aa-4bed-a7f4-517fb6923d2e-1510039682956';

window['beacon-url'] = 'http://192.168.9.12:7077';</script>

<script src='http://192.168.9.12:7077/rumcollector/egrum.js' async> </script>

<!-- RUM Header -->
```

- Then, save the script.
- Finally, inject the script into the web pages of the target web site / web application, using the appropriate instrumentation mechanism.

This way, you can separate the egrum.js from the collector, and thus make sure that the unavailability of the collector does not affect the web page loading time.

3.2 Installing and Configuring an eG RUM Collector

A default RUM collector (64-bit) 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) or WebLogic v10 (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 1.7 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 *eG Manager and Database Sizing Calculator*.
- The eG RUM Collector cannot be installed on a 32-bit operating system or on a 32-bit JDK/JRE.

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 3.3), 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. The browsers used by intranet and internet users will send the performance beacons to the collector via an HTTP/S connection to the collector's public URL. On the other hand, if the remote agent has been deployed on a host in the intranet, then the agent will poll only the private URL of the collector (via HTTP/S) to collect metrics.

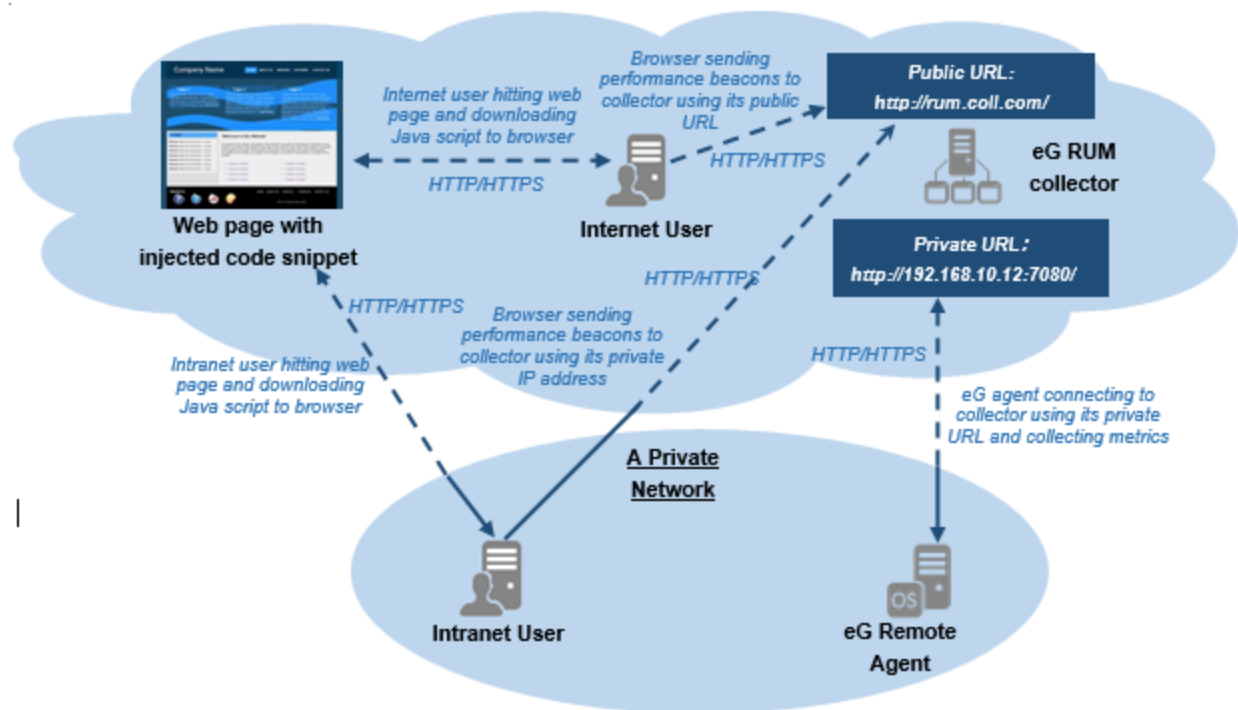


Figure 3.3: A RUM collector deployment model where the target web application is accessed by both internet and intranet users and the eG agent is in a private network

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 3.4). 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.

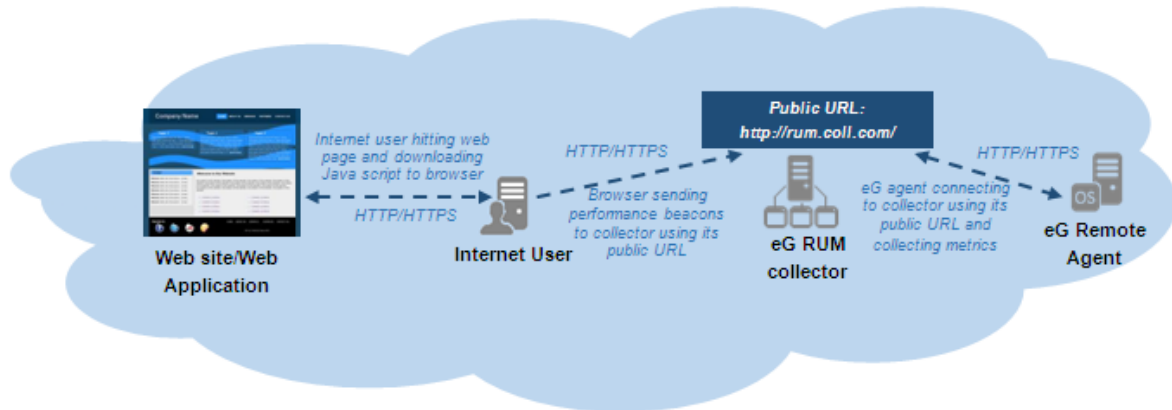


Figure 3.4: A RUM collector deployment model where both the target web application is used only by internet users and the eG agent is on the cloud

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 3.5). 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.

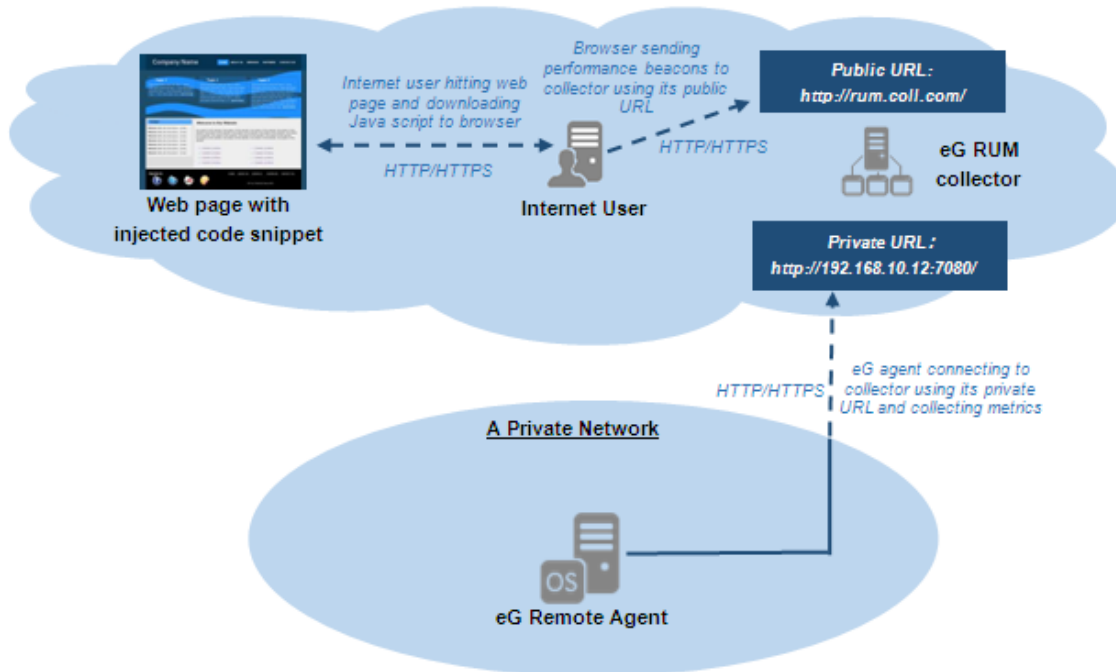


Figure 3.5: A RUM collector deployment model where the web application is used only by internet users and the eG agent alone is in a private network

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 3.6). 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.

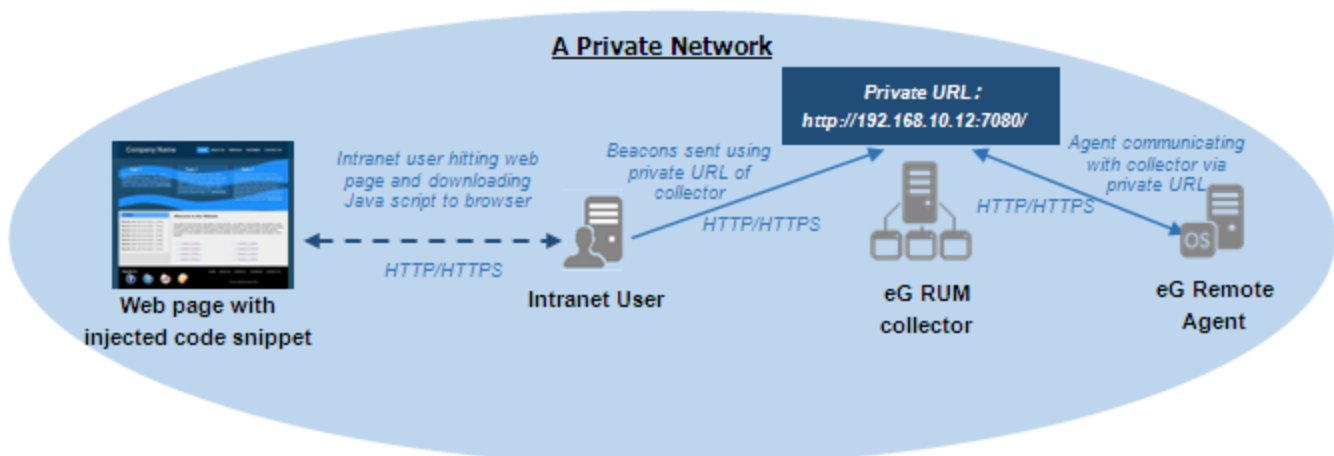


Figure 3.6: A RUM collector deployment model where the target web application is used only by intranet users and the eG agent is also in the same intranet

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 a public IP address alone (see Figure 3.7).

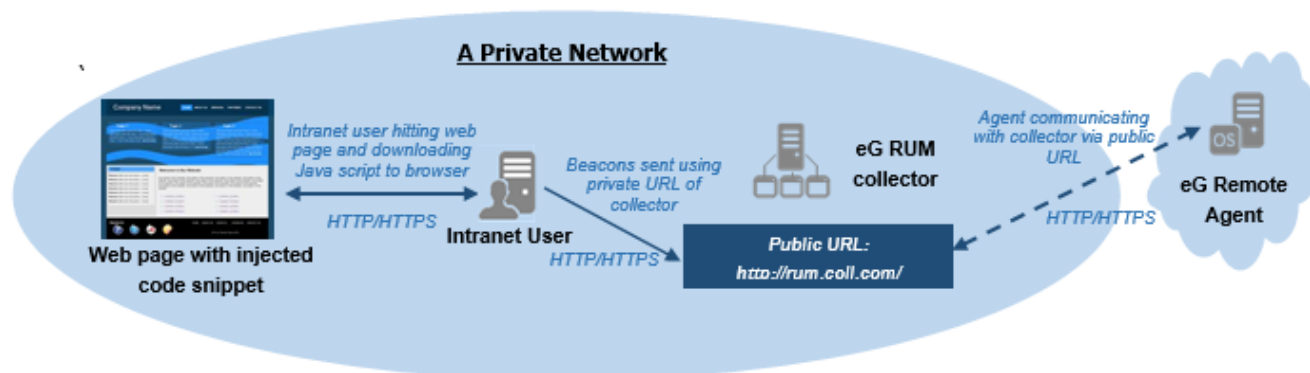


Figure 3.7: A RUM collector deployment model where the target web application is accessed by intranet users and the eG agent is on the cloud

3.2.3 Installing the eG RUM Collector

You can install the eG RUM Collector using one of the following methodologies:

- Using the **eGRUM** executable; (OR)
- Using the **rumcollector.war** file

Both these methodologies are described in the sub-sections below.

3.2.3.1 Installing Using the eGRUM Executable

The eG RUM collector is provided as a self-extracting, 64-bit program named **eGRUM**. This program can be executed only on a Windows 2008 (64-bit) or Windows 2012 (64-bit) host. This means that with the help of the **eGRUM** executable, you can install the RUM collector on a Windows 2008 (64-bit) or Windows 2012 (64-bit) host only.

To begin the installation, 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 3.8).

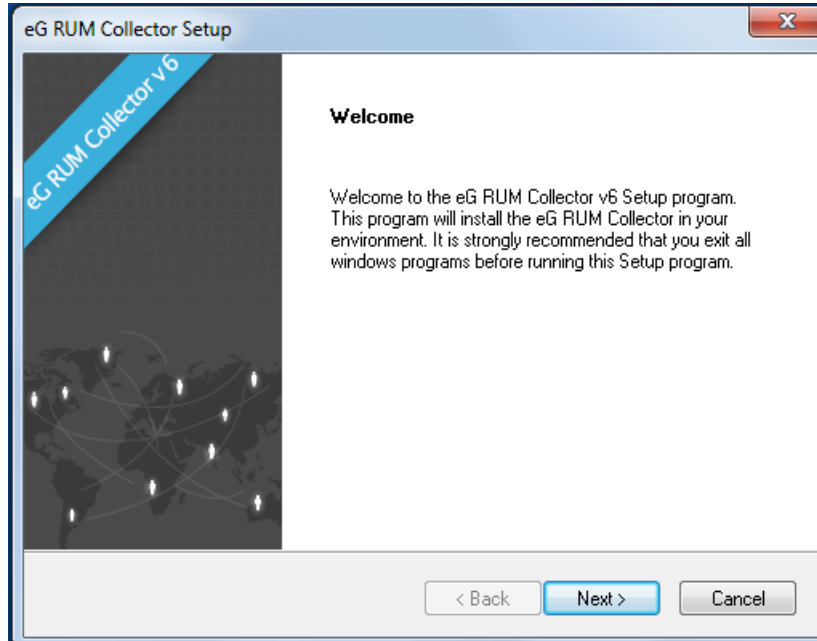


Figure 3.8: The Welcome screen

2. Click **Next** in Figure 3.8 to continue with the installation. Figure 3.9 will then appear.

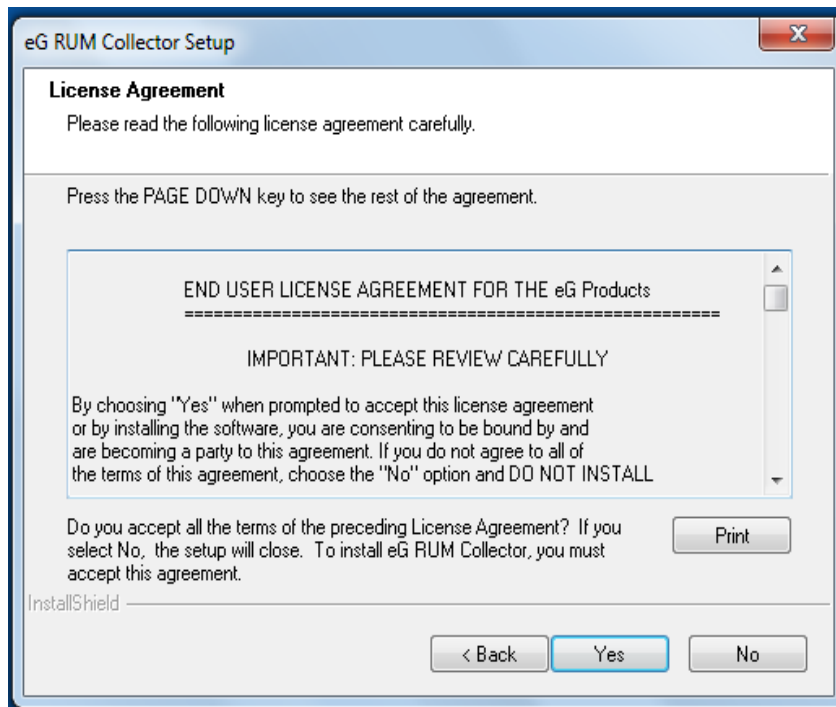


Figure 3.9: The End User License Agreement

3. Click the **Yes** button in Figure 3.9 to accept the terms of the license agreement. This will invoke Figure 3.10.

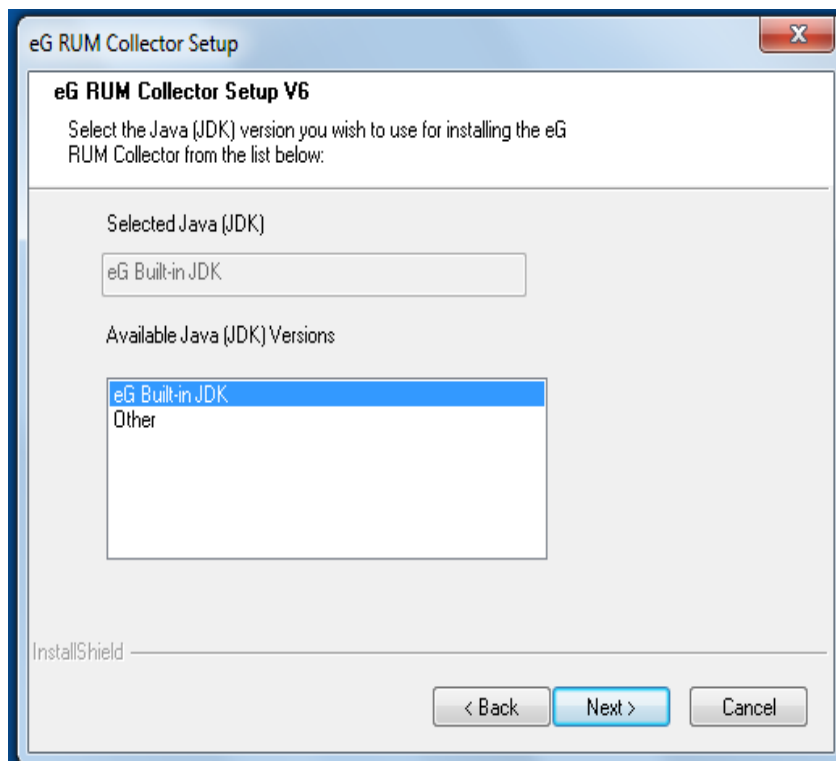


Figure 3.10: Selecting a JDK version to use for collector installation

4. Setup then automatically discovers all the versions of JDK available on the target host, and lists them as depicted by Figure 3.10. 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 1.7 (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 3.10 for some reason, he/she can pick the **Other** option from Figure 3.10. Figure 3.11 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.**

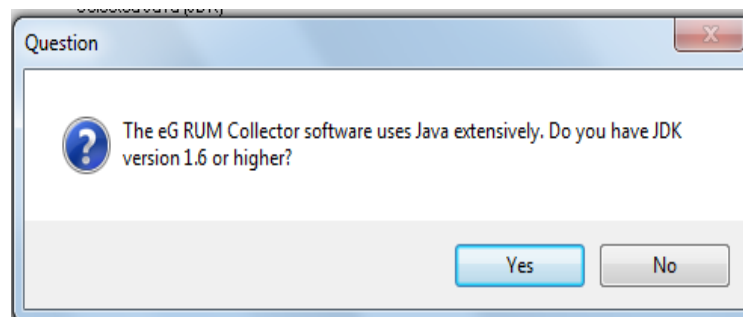


Figure 3.11: Setup enquiring the availability of JDK in the environment

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 3.12. The user can also use the **Browse** button to select the location of the Java home directory.

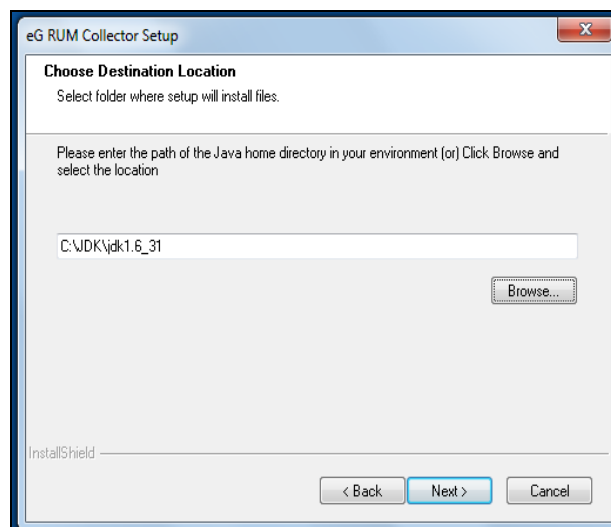


Figure 3.12: Specifying the location of the Java home directory for installing the eG RUM collector

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 3.13). 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 3.13. 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 3.13. This will invoke Figure 3.14 where you can manually specify 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.

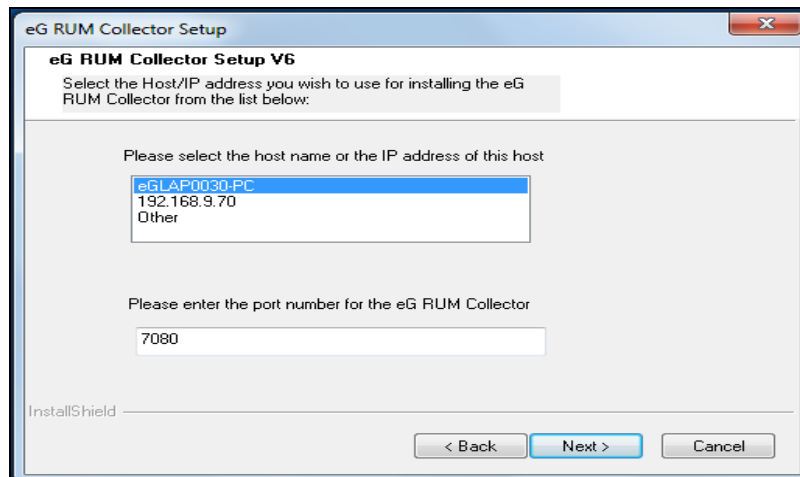


Figure 3.13: Selecting the IP address/host name of the collector

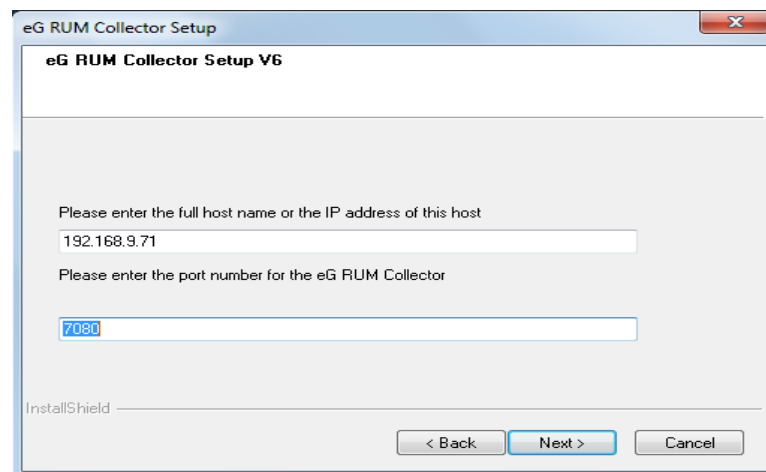


Figure 3.14: Manually specifying the IP address/host name of the collector host

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 3.15. If not, click **No**.

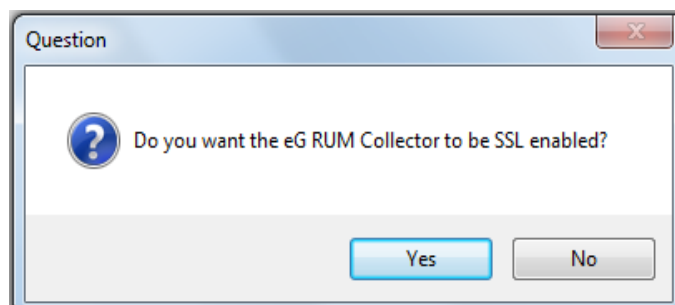


Figure 3.15: Indicating whether/not to SSL-enable the eG RUM collector

10. Next, use the **Browse** button in Figure 3.16 that appears to specify where the eG RUM collector is to be installed. Then, click **Next** to proceed.

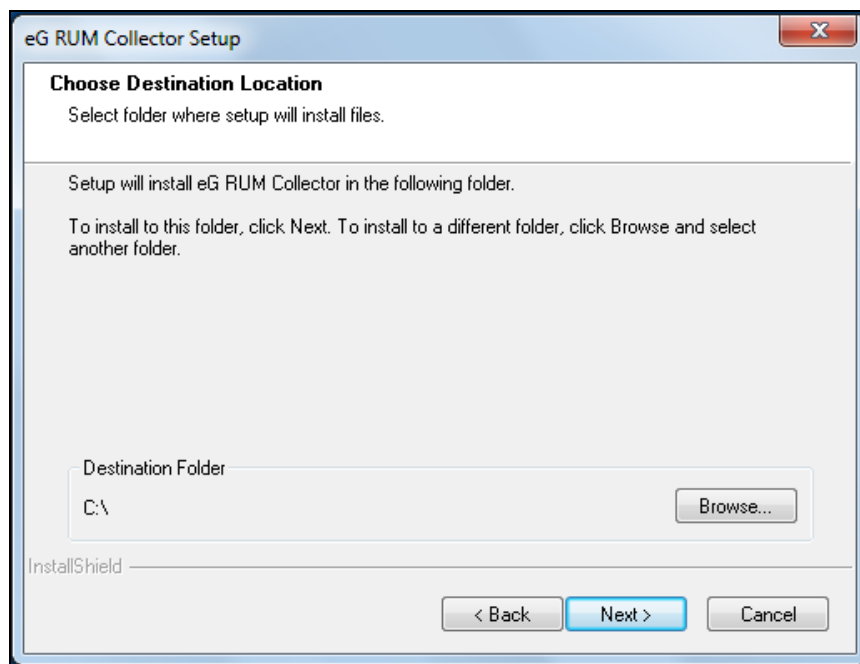


Figure 3.16: Specifying the install directory of the eG RUM collector

11. Then, view a quick summary of the specifications and click **Next** in Figure 3.17 to proceed.

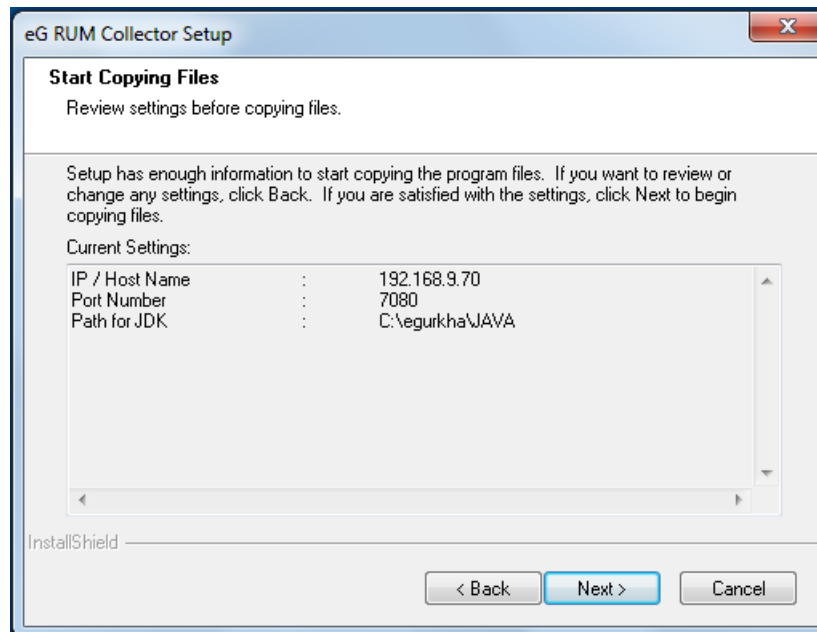


Figure 3.17: A summary of the specifications

12. If the configuration process succeeds, the following screen will be displayed (see Figure 3.18). 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.

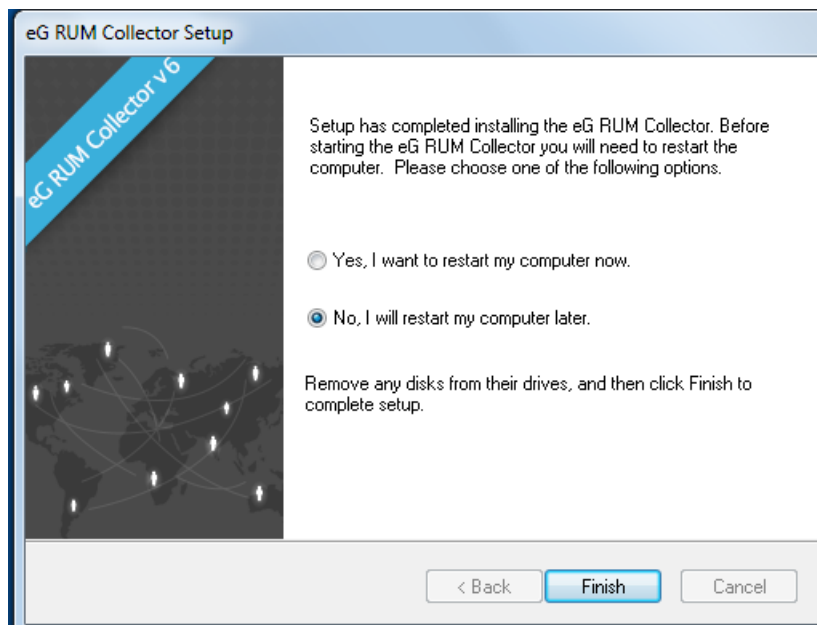


Figure 3.18: Setup program indicating the completion of the eG RUM collector installation

Since the eG RUM collector is a Tomcat-based application, the **eGRUM** executable is bundled with Apache Tomcat v6. An Apache Tomcat server therefore, gets automatically installed on the target Windows host when deploying the collector.

3.2.3.2 Installing Using the WAR File

The eG RUM collector is also available as a war file named **rumcollector.war** in the **/opt/egurkha/manager/tomcat/webapps** (on Unix; on Windows, this will be the <EG_INSTALL_DIR>\manager\tomcat\webapps) folder of the eG manager. This war file can either be installed on an Apache Tomcat server v6 (or above) or on an Oracle WebLogic server v10.x/11.x/12.x.

To deploy the war file on a Tomcat server, 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 the 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.

To deploy the rumcollector.war file on a WebLogic server, do the following:

1. Login to the target WebLogic server.
2. Copy the **rumcollector.war** file from the eG manager host to any location on the WebLogic server.
3. Launch a browser and then connect to the WebLogic console.
4. Login to the console as a WebLogic administrator.
5. Figure 3.19 will then appear.

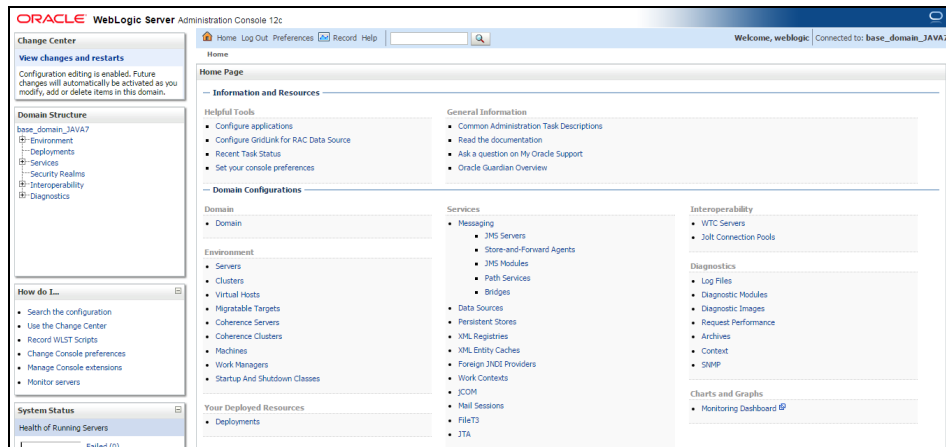


Figure 3.19: The WebLogic console

6. Click on the **Deployments** node in the **Domain Structure** tree in the left panel of Figure 3.19. Figure 3.20 will then appear.

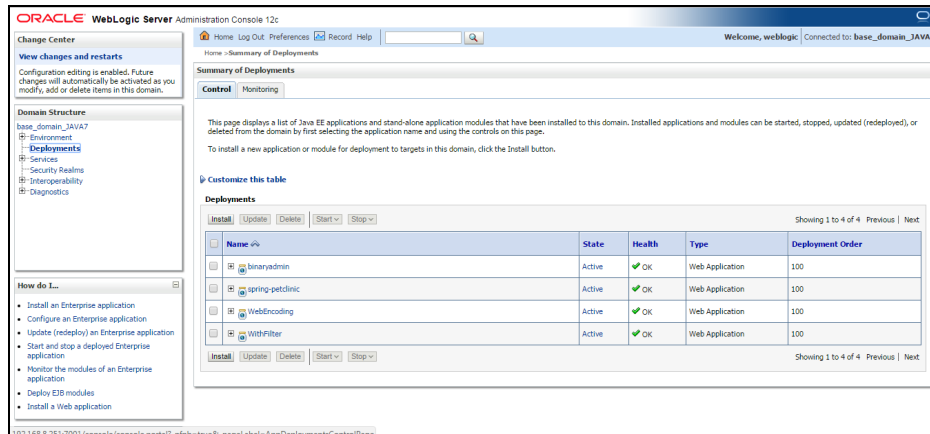


Figure 3.20: List of applications deployed on the server

- To install the eG RUM Collector web application, click on the **Install** button in the right panel of Figure 3.20. This will open Figure 3.21.

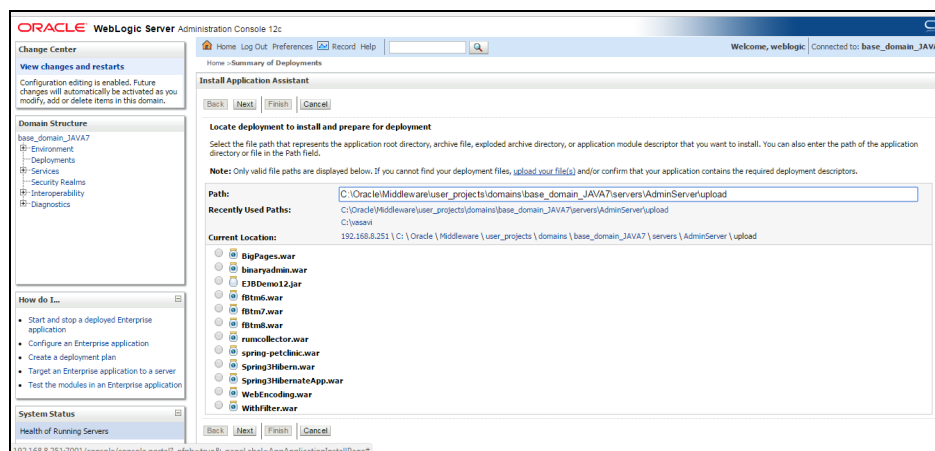


Figure 3.21: Clicking the 'upload your file(s)' link

- Click on the **upload your file(s)** link against **Note:** in the right panel of Figure 3.21. Figure 3.22 will then appear, where you have to choose the rumcollector.war file that you wish to upload. For that, use the button **Choose** in Figure 3.22.

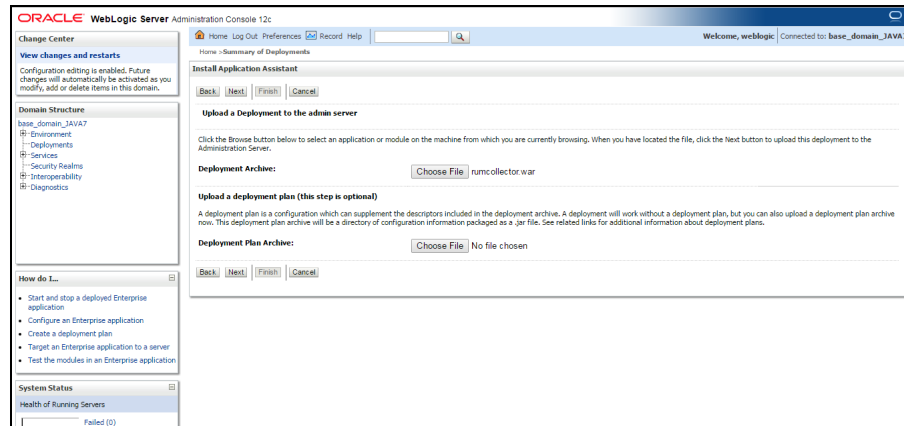


Figure 3.22: Choosing the rumcollector.war file to be uploaded

- Then, click the **Next** button in Figure 3.22 to proceed. If the upload is successful, Figure 3.23 that appears next will display a message to that effect. Click **Next** in Figure 3.23 to proceed.

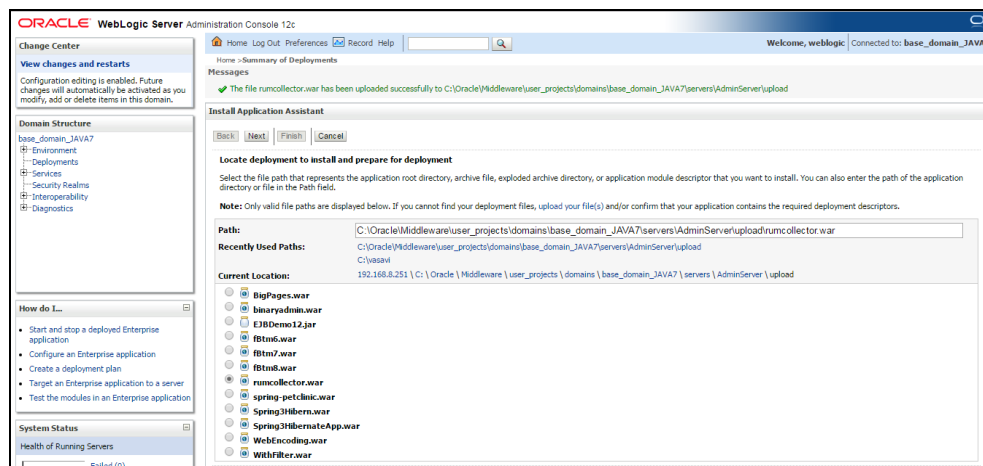


Figure 3.23: Message confirming the successful file upload

- When Figure 3.24 appears, click the **Next** button again.

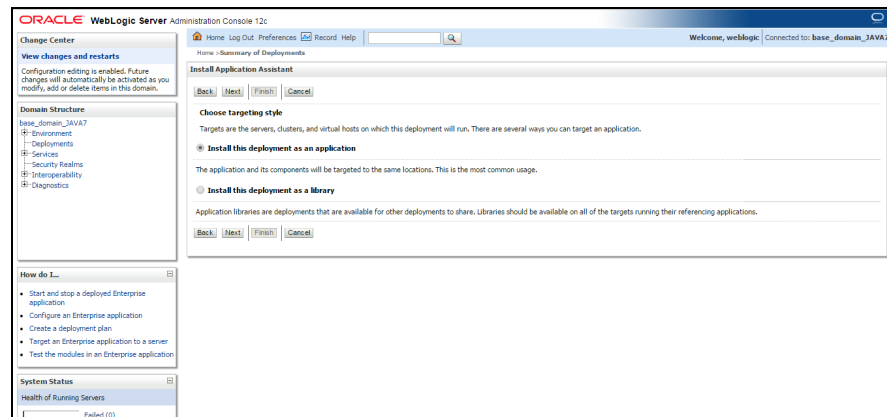


Figure 3.24: Clicking the Next button to proceed

- Click the **Finish** button in Figure 3.25 to end the war file installation.

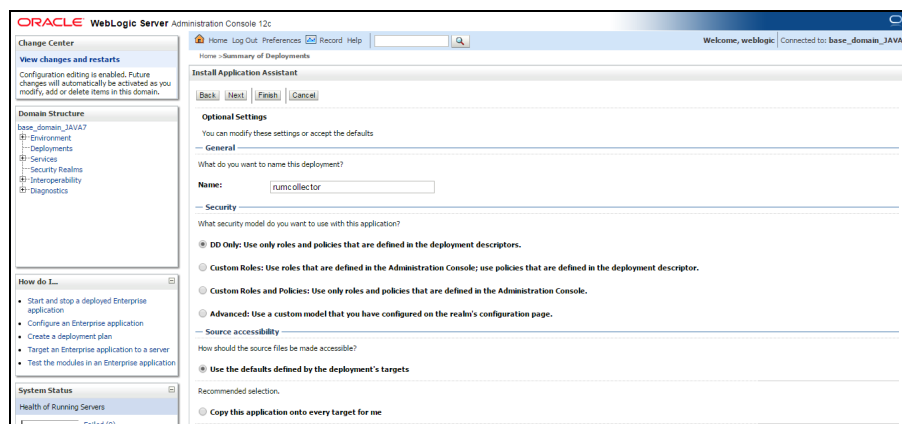


Figure 3.25: Clicking Finish to end the war file installation

- Figure 3.26 then appears. If the `rumcollector.war` has been deployed successfully, then, in the **Deployments** list of Figure 3.26, you should be able to find an entry for the `rumcollector`, with **State** as *Active* and **Health** as *OK*. If you find such an entry, you can confirm the successful installation of the eG RUM Collector.

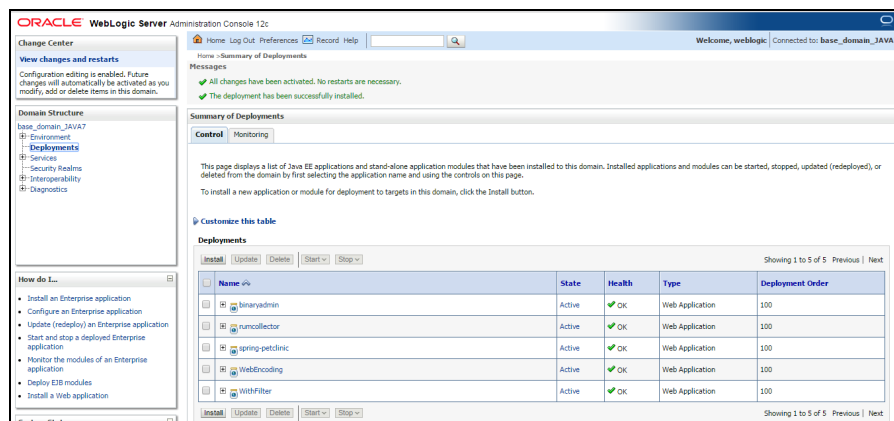


Figure 3.26: Successful installation of the eG RUM Collector

13. Finally, log out of the WebLogic console.

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, **you should ideally SSL-enable the eG RUM collector**. This is because, the basic security framework of browsers does not permit browsers to send resources and beacons from an HTTPS web site/application to an HTTP collector.

For ease of management, you can front the HTTP collector with an SSL-enabled load balancer (eg., F5, NetScaler, etc.). This load balancer can then be configured to forward the resources and beacons it receives to the HTTP collector.

If you do not want to use a load balancer, then you will have to SSL-enable the RUM collector.

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 3.15) at step 9 of the collector installation process detailed in Section 3.2.3.1 . **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 certifying 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. If you pick option (b), use the procedure detailed in Section 3.2.5.2 .

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 public-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/certificate 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).
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 3.27 will then appear.

The screenshot shows a web browser window titled "Microsoft Active Directory Certificate Services - Windows Internet Explorer". The address bar shows "http://10.12.2.34/certsrv/certreq.asp". The page title is "Microsoft Active Directory Certificate Services - agcut-AGC1UAT-CA". The main heading is "Submit a Certificate Request or Renewal Request". Below this, a paragraph states: "To submit a saved request to the CA, paste a base-64-encoded CMC or PKCS #10 certificate request or PKCS #7 renewal request generated by an external source (such as a Web server) in the Saved Request box."

The "Saved Request:" section contains a large text area with a dropdown menu. The dropdown is set to "Base-64-encoded certificate request (CMC or PKCS #10 or PKCS #7)". The text area contains the following text:

```
038xbB68Y0pgep91LZdv1Au62aJp+Frv1a6Tgx2O
ab66Iuc1ypc2k2685fch1p01R08y0cTIApdyCQaZ
3ond3Uw1DQaBu&W0QY3R0Z1Iwv0hA0EP0Q0hTEA
085ux853+52c8K8K0t1G0/AlTDe1Dv4Hw0q7EAm
eB/Bw3ut1K9p7T1h0/y661P1Mh2EPPK/1ux5tc8u
-----END NEW CERTIFICATE REQUEST-----
```

Below the text area is a "Certificate Template:" section with a dropdown menu set to "Web Server". Below that is an "Additional Attributes:" section with a label "Attributes:" and a text area. At the bottom right of the form is a "Submit >" button.

Figure 3.27: 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), 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 3.27. Then, click the **Submit** button.
3. The certificate will thus be generated. Download the certificate.

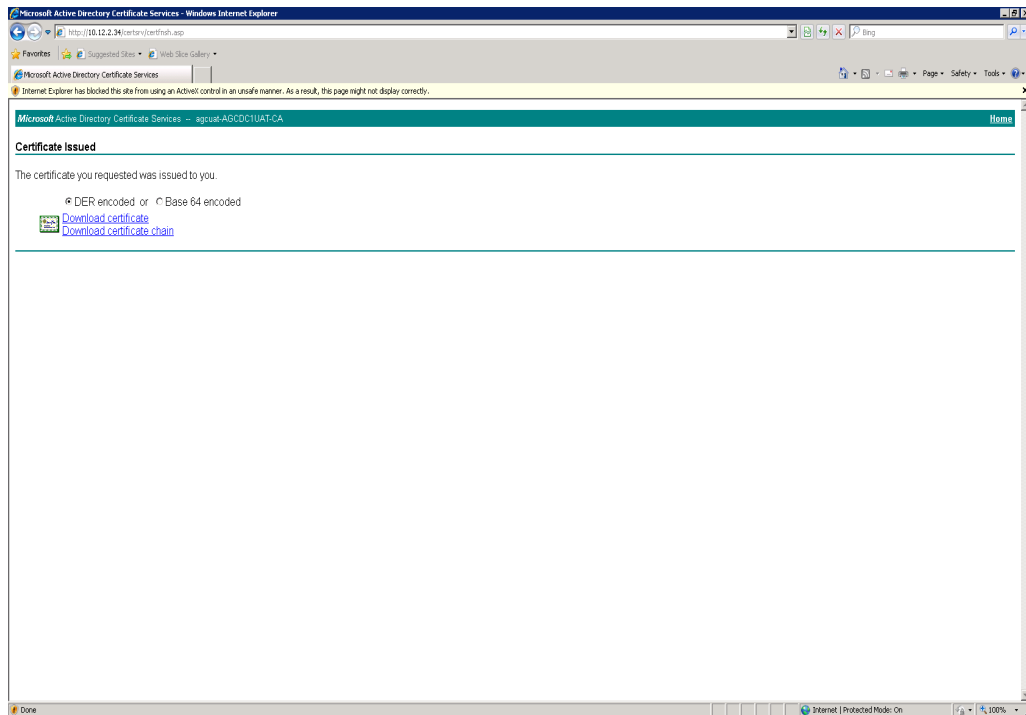


Figure 3.28: Downloading the certificate

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 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**.
 - **-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 -keypassmykey -storepassmykey
```

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 **3.2.5.1.1** 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.
- **-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

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"
    useURValidationHack="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"/>
```

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"
useURValidationHack="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 using the procedure detailed in the Section 3.2.5.1.1 topic.

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"  
    useURIVValidationHack="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 public-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/certificate 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**).
 - **-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**). 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**).
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 3.2.5.1.1.
 - **-keypass** and **-storepass** : Provide the same **keypass** and **storepass** that you specified when generating the keystore; refer to Section 3.2.5.1.1.
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 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.

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.

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.
- **-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

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"
    useURIVValidationHack="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"/>
```

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"
useURIVValidationHack="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.

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"
useURIVValidationHack="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:

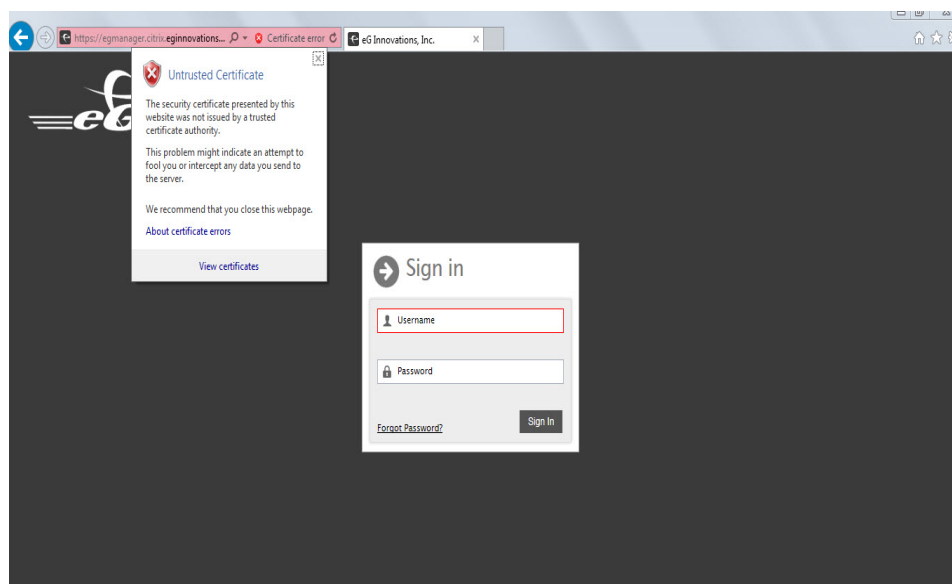


Figure 3.29: The “Certificate error” that the browser reports

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 3.30) to import the certificate to the browser.

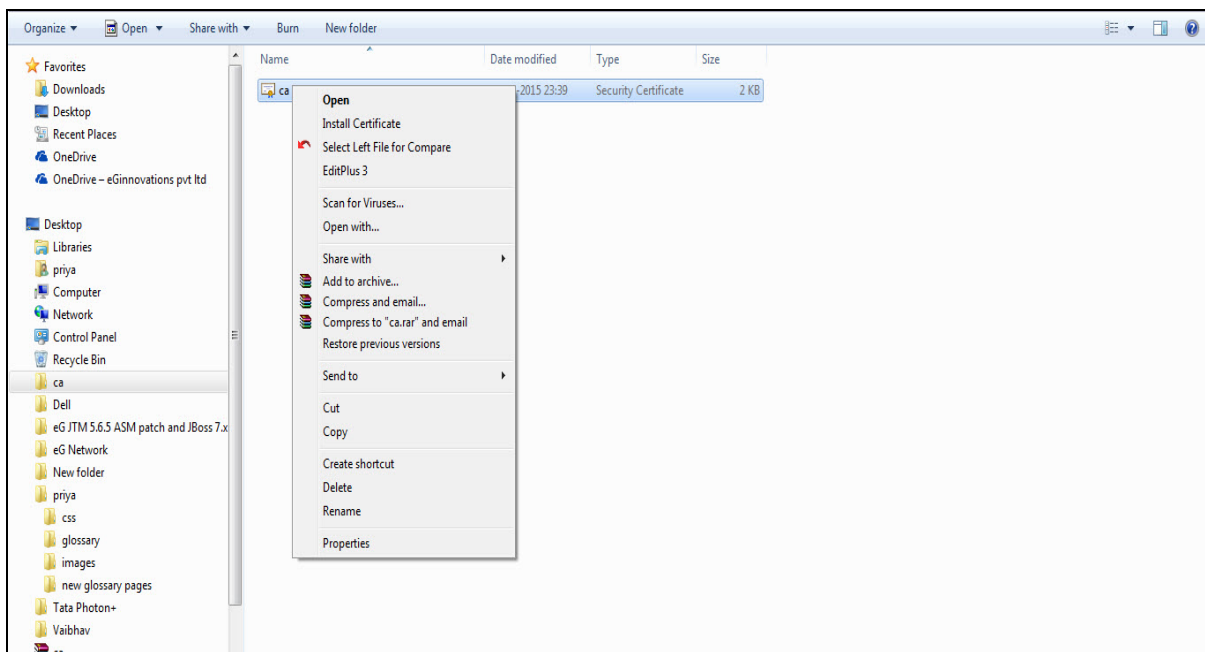


Figure 3.30: Selecting the option to install the certificate on the browser host

3. Figure 3.31 will then appear. Click **Next** here to continue.



Figure 3.31: Welcome screen of the Certificate Import Wizard

4. Figure 3.31 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.

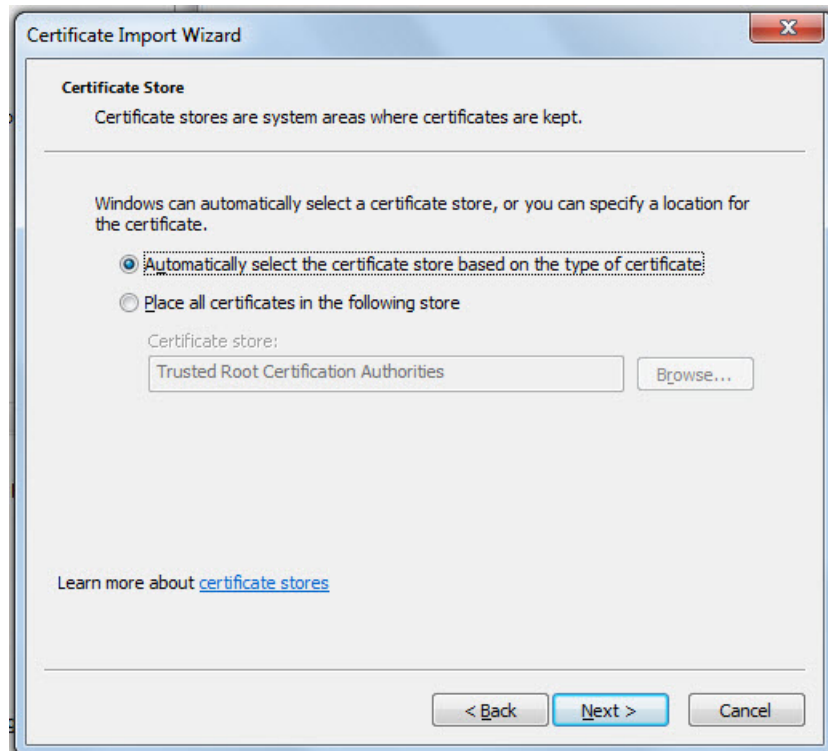


Figure 3.32: Choosing to place the certificate in a specific store

5. From Figure 3.33 that then appears, select the **Trusted Root Certificate Authorities** store and click **OK**.

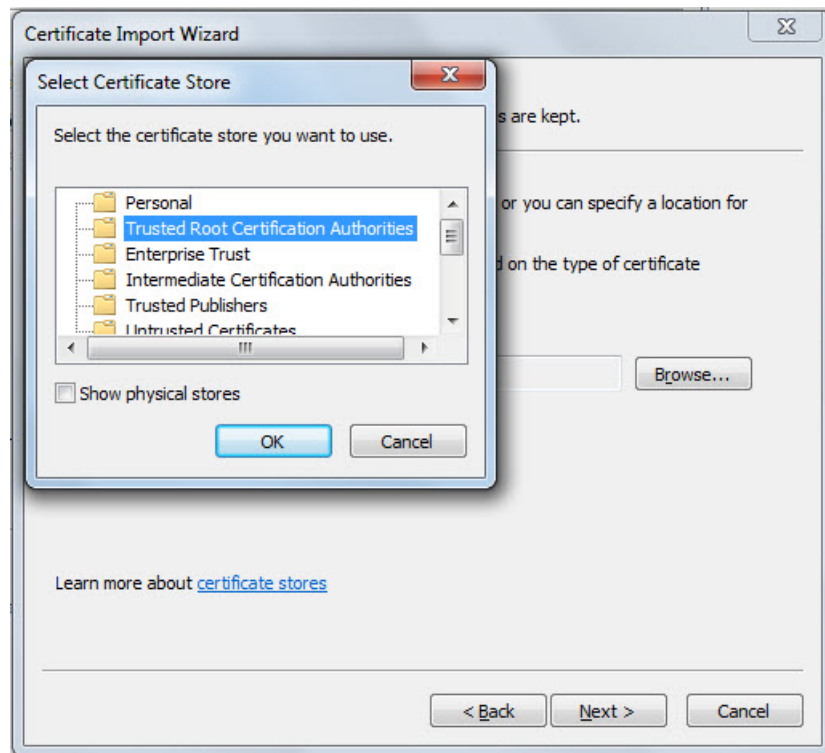


Figure 3.33: Storing the certificate in the Trusted Root Certificate Authorities store

6. The chosen store will then appear in the text box below **Place all certificates in the following store** option, as depicted by Figure 3.34. Click **Next** in Figure 3.34 to continue.

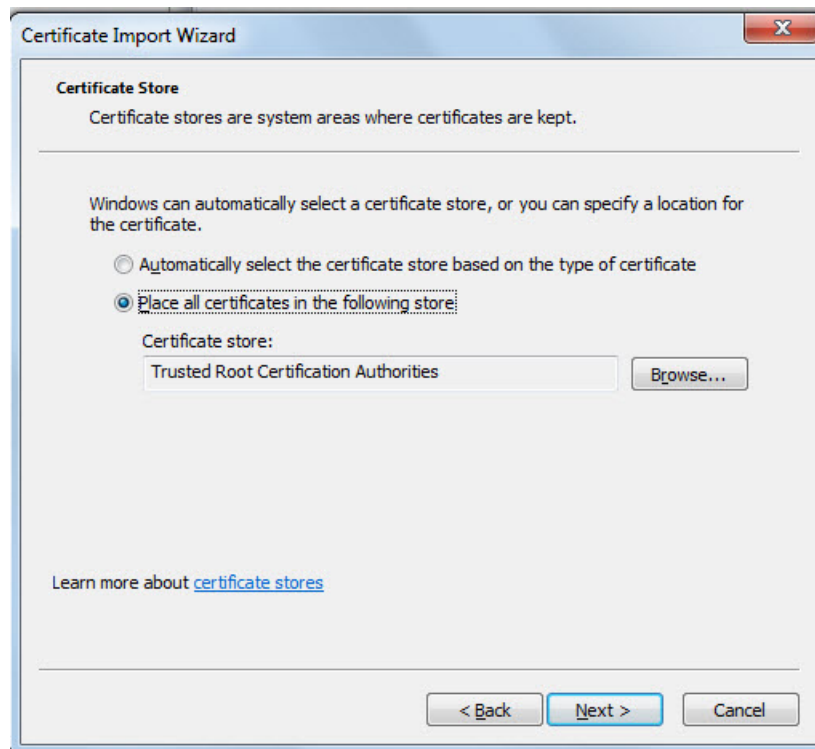


Figure 3.34: The chosen store displayed

7. A quick summary of your selections will appear in Figure 3.35. Review your specifications and click **Finish** to complete the import.

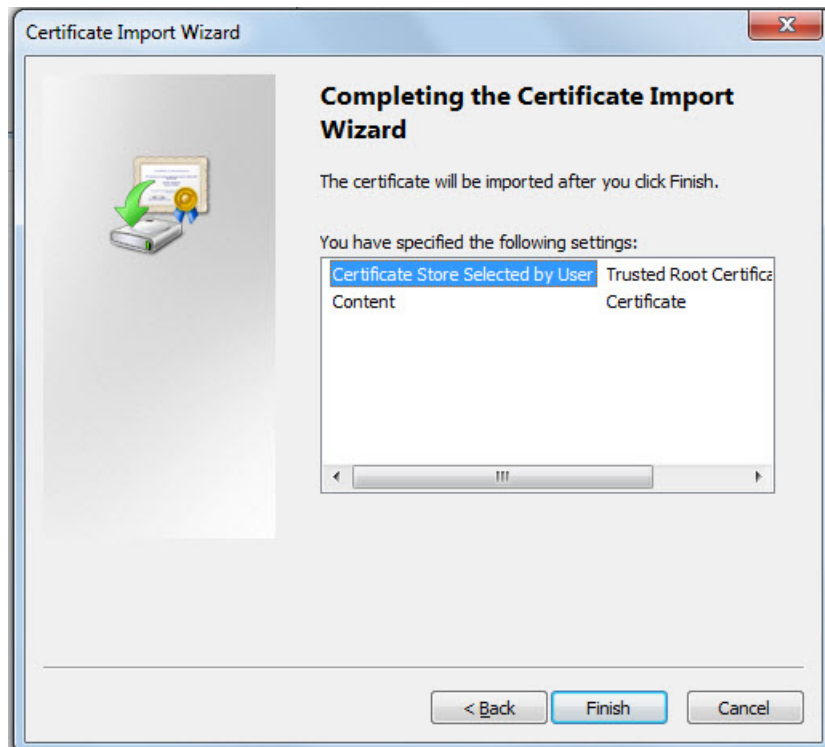


Figure 3.35: Finishing the import

8. The following warning message will appear. Click **Yes** in Figure 3.36 to proceed with the import.

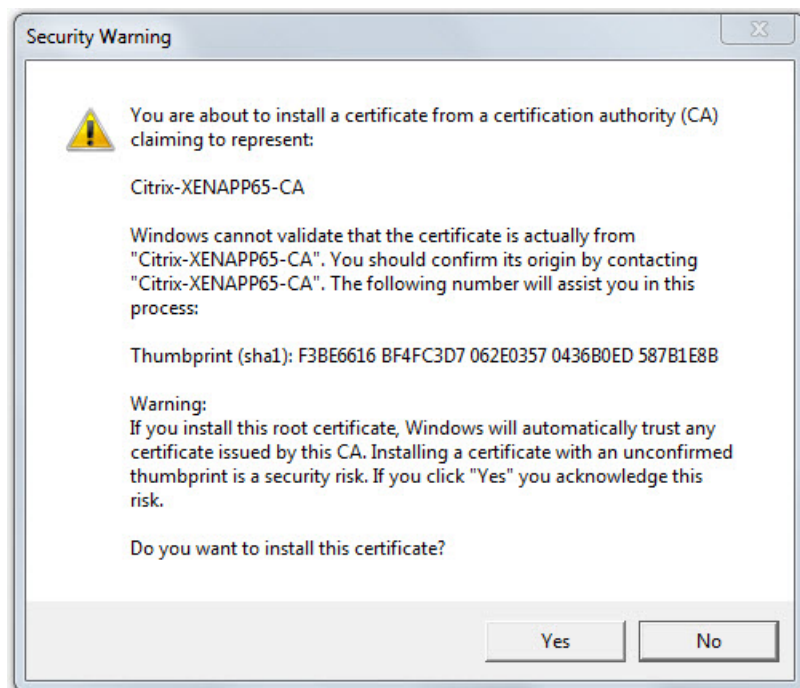


Figure 3.36: A warning message that appears when importing a certificate issued by an internal CA

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

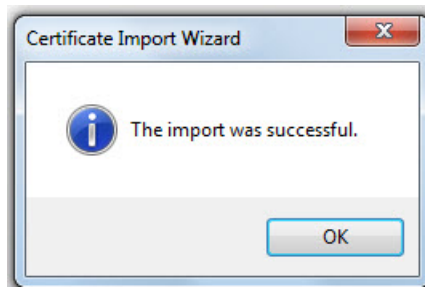


Figure 3.37: A message box informing you that the certificate has been successfully imported

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

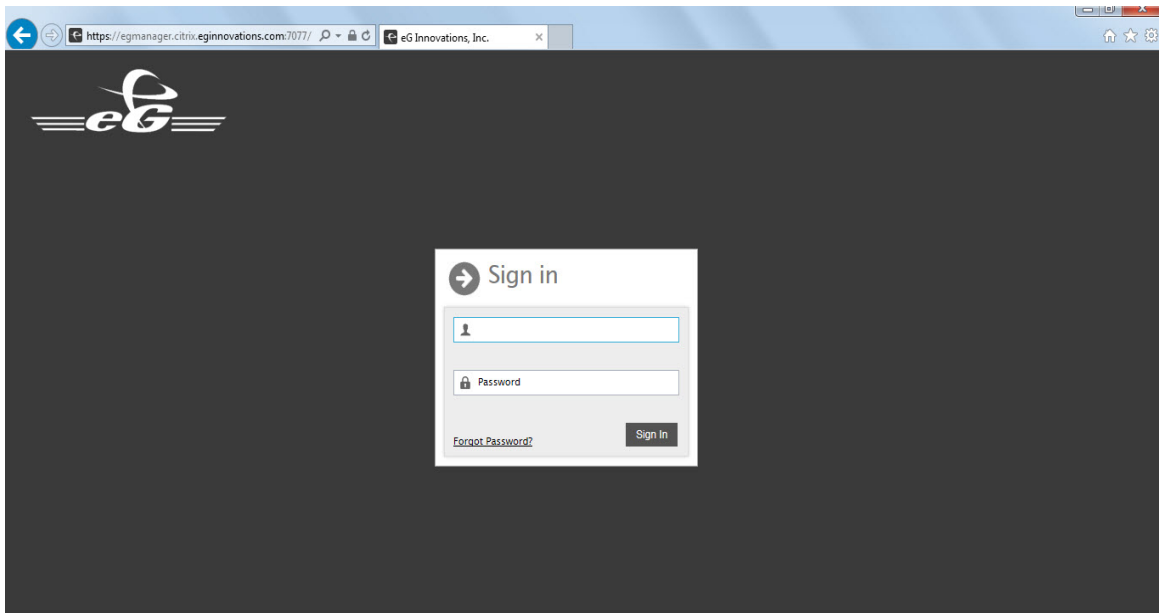


Figure 3.38: The login screen of the eG manager, without the 'Certificate error'

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 collectors** options from the **Miscellaneous** tile of the **Admin** tile menu.
3. Figure 3.39 will then appear listing the collectors that pre-exist.

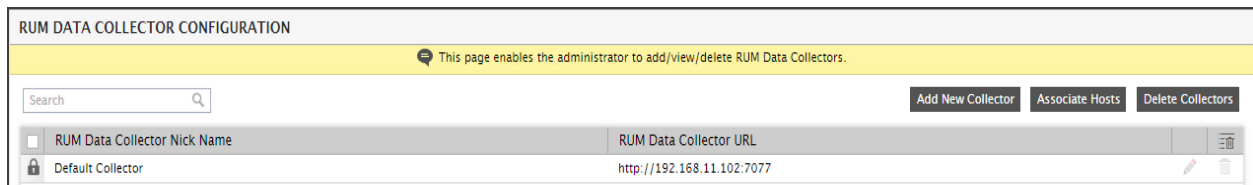


Figure 3.39: List of RUM collectors that pre-exist

4. The eG manager is bundled with a default RUM collector. By default, this default collector will be listed in Figure 3.39. 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 3.39.
6. Figure 3.40 then appears. In the **Nick Name** text box of Figure 3.40, 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 3.40: 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 3.40 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 3.41 will then appear.

Figure 3.41: Choosing to add a Real User Monitor

3. Select **Real User Monitor** as the **Component type** from Figure 3.41 and click the **Add New Component** button to add a new component of that type.
4. When Figure 3.42 appears, provide a unique name for the web site/web application that you

want to monitor in the **Nick name** text box.

Component Information	
Category	All
Component type	Real User Monitor
Nick name	MyWebSite_RUM
RUM collector	Default Collector
Remote agent	192.168.8.26
Capture UserName	☐ Yes ☒ No
Add	

Figure 3.42: 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 3.40) of each of the RUM collectors that you have configured in your environment will populate the **RUM collector** drop-down in Figure 3.42. 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. The detailed diagnostics provided by the eG Real User Monitor reveal the exact URLs in the target web site/web application that are slow to respond to user requests. By default however, these detailed metrics do not provide the names of the users who hit the slow URLs. If you want to include the user name in detailed diagnosis, then set the **Capture UserName** flag to **Yes** (default: **No**). Once the flag is turned on, follow the steps detailed in the Enabling the eG Real User Monitor to Capture User Names topic, so you can easily configure eG RUM to capture the user name from a slow-loading web page and report it as part of detailed diagnosis. With the help of such information, help desk can perform troubleshooting more effectively. For instance, help desk can quickly identify the users who are experiencing slowness, well before they actually complain.
8. Then, click the **Add** button in Figure 3.42 to add the *Real User Monitor* component. Doing so will invoke Figure 3.43. Figure 3.43 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.

To instrument your html pages use the eG javascript agent

Download this if you have a Java web application that emits HTML to the front-end. Examples: JSF, JSP, Groovy: [egrum.jar](#) Download

Download this if you have a .NET web application such as ASP.NET MVC or WebForms: [Eg.RUM.dll](#) Download

Download this if you can edit the source code of your web pages. Examples: Plain HTML, JSP with embedded HTML, PHP with embedded HTML: [egrum.js](#) Download

Download this if want to embed this egrum.js into packaged applications such as SAP Fiori: [Code snippet + egrum.js](#) Download

Select the source file type on which eG RUM JavaScript code snippet to be included

Include this line in your pages (html/jsp/asp) immediately after the head tag

☒ HTML ☐ Javascript

```
<!-- RUM Header -->
<script charset='UTF-8' type='text/javascript'> window['egrum-start_time'] = new
Date().getTime();
window['Site_Name'] = '37951ab5-af89-499b-85c5-b6bd240a153c-1575469485811';
window['beacon-url'] = 'https://192.168.8.26:7077';
</script>
<script src='https://192.168.8.26:7077/rumcollector/egrum.js' async> </script>
```

Figure 3.43: Viewing the code snippet to be injected into the monitored web pages

3.4 Instrumenting the Web Site / Web Application to be Monitored

Figure 3.44 clearly delineates the options available for RUM-enabling a web site/web application, and when each of those options become applicable.

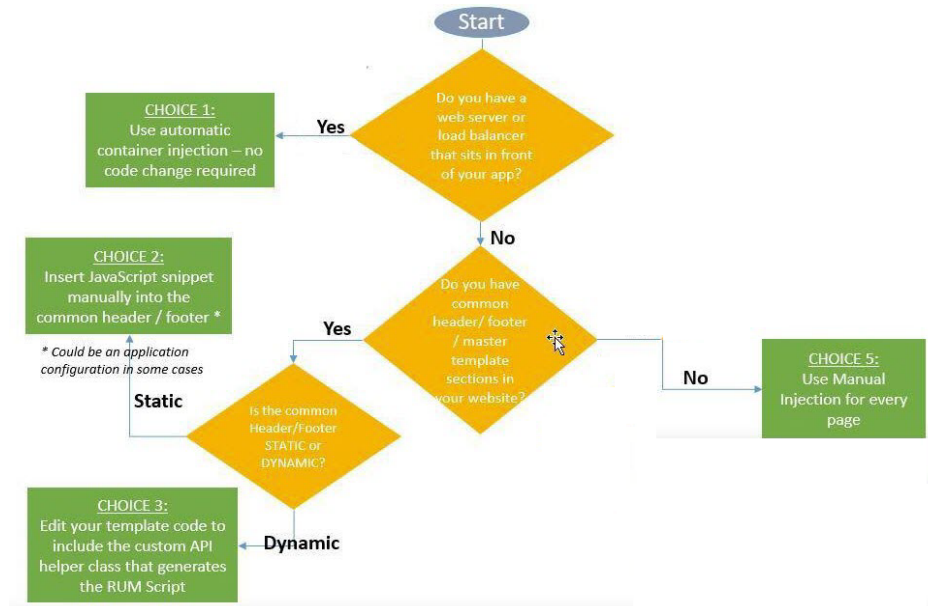


Figure 3.44: The options available for RUM-enabling a web site / web application and when those options become applicable

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, IBM HTTP server or Oracle HTTP server) or is fronted by a load balancer, 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, IBM HTTP, and an Oracle HTTP server, this can be achieved by following the steps detailed in Section 3.4.2, Instrumenting an IBM HTTP Server v9.x (and above) Using Container-Assisted Injection and Section 3.4.4 topics. Where load balancers such as NetScaler and F5-Big IP front-end the target web site/web application, follow the procedure detailed in the Section 3.4.10 topic and the Section 3.4.11 topic respectively to inject the RUM script.

Note:

Container-assisted RUM script injection can be used to RUM-enable only those web servers that support the *mod_substitute* and *mod_filter* modules. If a web server does not support these modules or supports only one of them, then such web servers cannot be RUM-enabled using the container-based injection method. To RUM-enable such web servers, any of the alternate approaches depicted by Figure 3.44 should only be used.

- **Adding JavaScript code to common header/footer/include section:** If a web site / web application is **not** hosted on IIS, Apache, or any variants, check whether that web site / application has a common header/footer/include section. If it does, then check if the common section is editable. If so, then do either of the following:
 - Manually insert the JavaScript code into the common header/footer/include section; this way, the JavaScript will get automatically injected into each web page that uses that common header/footer/include section;
 - Manually add the JavaScript code into the 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 3.4.7 and Section 3.4.8 topics.

On the other hand, if the common section is not editable, then check whether the target application uses server-side templates to auto-generate the common section. If this is the case, then edit the template code to include the custom API helper class that generates the RUM script. The following helper classes are available:

- **RUM Library for Java:** Only Java-based web sites / web applications can be instrumented using this option. 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 the template code that auto-generates the common section, and add the code block that eG prescribes to that template. To know what code block needs to be inserted in the template, refer to the Section 3.4.5 topic. This code block will automatically inject a JavaScript into the common section that is auto-generated from that template. This JavaScript will then be automatically injected into each of the web pages that use that common section. This script will track the requests to that web page, capture the response time metrics of these requests, and send these metrics to the collector. For the detailed procedure, refer to the Section 3.4.5 topic.

- **RUM Library for DotNet: Only DotNet-based web sites / web applications can be instrumented using this option.** 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, you will have to edit the **Views\Shared_Layout.cshtml** or **Default.master** file and insert the code block described in the [Instrumenting a DotNet-based Web Site / Web Application](#) topic in it. 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 3.4.6 topic.
- **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 **does not support common header/footer/include section**. 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 3.4.13 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 compression (static and/or dynamic) 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 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 3.45), and then select the Compression option from the right panel.

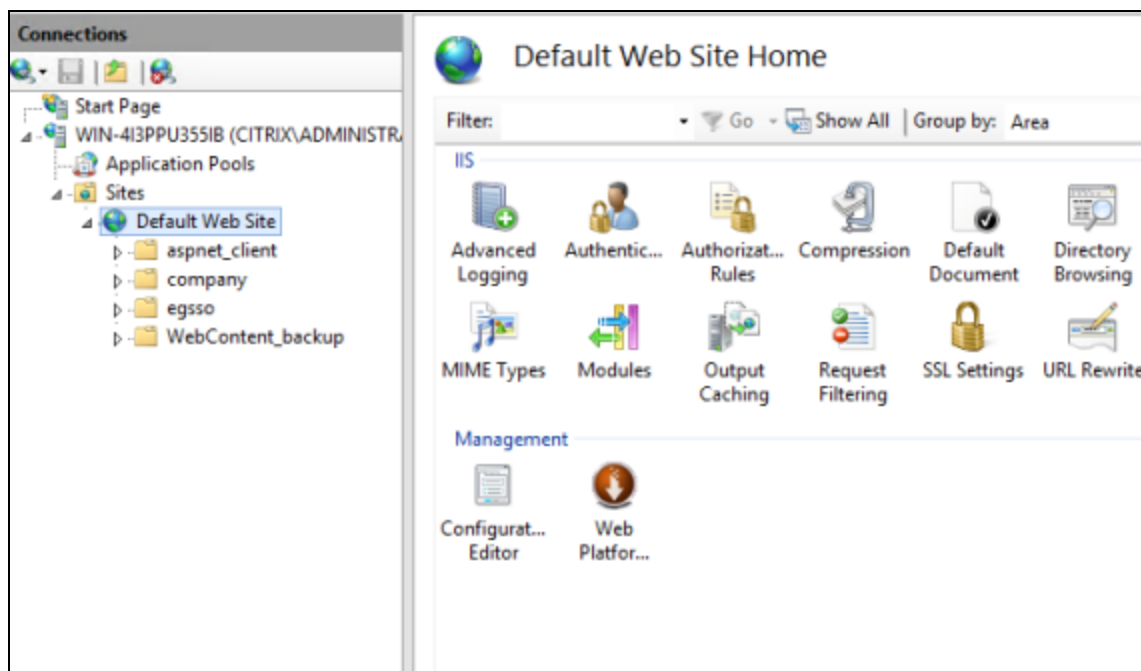


Figure 3.45: Accessing the Compression menu

- Then, when Figure 3.46 appears, check whether the Enable static content compression check box and/or the Enable dynamic content compression check box is selected; if so, it means that Compression has been enabled.

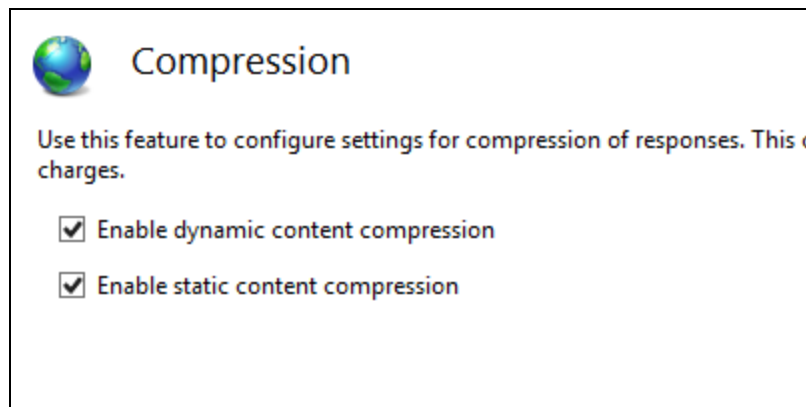


Figure 3.46: Checking whether static compression is enabled

If either or both the check boxes in the Compression section is selected in Figure 3.46, 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
```

If the target web application is a 32-bit application that is running on a 64-bit machine, then execute the following command instead:

```
reg add HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432node\Microsoft\Inetstp\Rewrite /v  
LogRewrittenUrlEnabled /t REG_DWORD /d 0
```

[Click here](#) to know how to verify whether a target web application is a 32-bit application or a 64-bit application.

- Next, make sure that the dynamicCompressionBeforeCache property is set to false for the /system.webServer/urlCompression configuration element. For this, add the following entry in the web.config file as depicted by 3.4.1.

```
<urlCompression doStaticCompression="false" doDynamicCompression="true"  
dynamicCompressionBeforeCache="false" />
```

```

<system.webServer>
  <urlCompression doStaticCompression="false" doDynamicCompression="true" dynamicCompressionBeforeCache="false" />
  <tracing>
    <traceFailedRequests>
      <add path="*" />
      <traceAreas>
        <add provider="WWW Server" areas="WebSocket" verbosity="Verbose" />
      </traceAreas>
      <failureDefinitions statusCodes="200-502" />
    </add>
  </traceFailedRequests>
</tracing>
<rewrite>
  <outboundRules>
    <rule name="eG RUM Head Rule" preCondition="isHtmlContent">
      <match filterByTags="None" pattern="<!-->" />
      <action type="Rewrite" value="{R:0}<script> window["egrum-start_time"] = new Date().getTime(); </script>" />
    </rule>
    <rule name="eG RUM Body Rule" preCondition="isHtmlContent">
      <match pattern="</body>" />
      <action type="Rewrite" value="<script> window["Site_Names"] = "egm_Sites"; window["beacon-url"] = "http://xyz.com";</script> <script src="http://xyz.com/rumcollector/egrum.js" </script>{R:0}" />
    </rule>
    <preConditions>
      <preCondition name="isHtmlContent">
        <add input="{RESPONSE_CONTENT_TYPE}" pattern="text/html" />
      </preCondition>
    </preConditions>
  </outboundRules>
</rewrite>

```

Figure 3.47: Setting the dynamicCompressionBeforeCache property to false

Alternatively, you can run the following command from the command prompt:

```
appcmd.exe set config "WEBSITENAME" -section:system.webServer/urlCompression
/dynamicCompressionBeforeCache:"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.
- Restart IIS.

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 3.48).

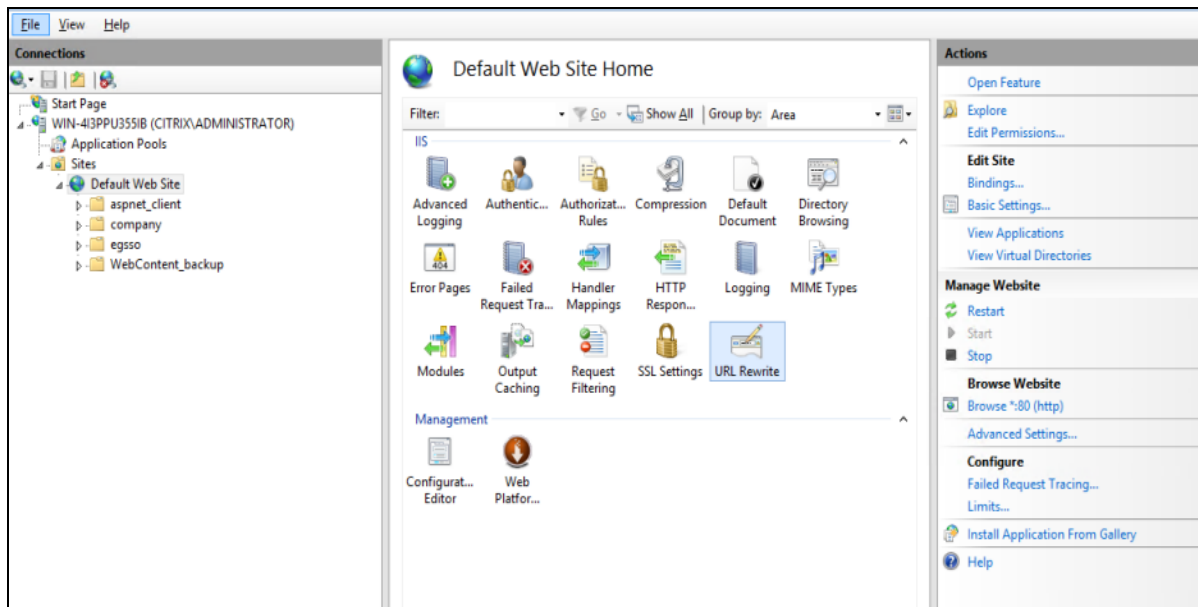


Figure 3.48: Accessing the URL Rewrite option

- When Figure 3.49 appears, select the **Add Rule(s)** option.

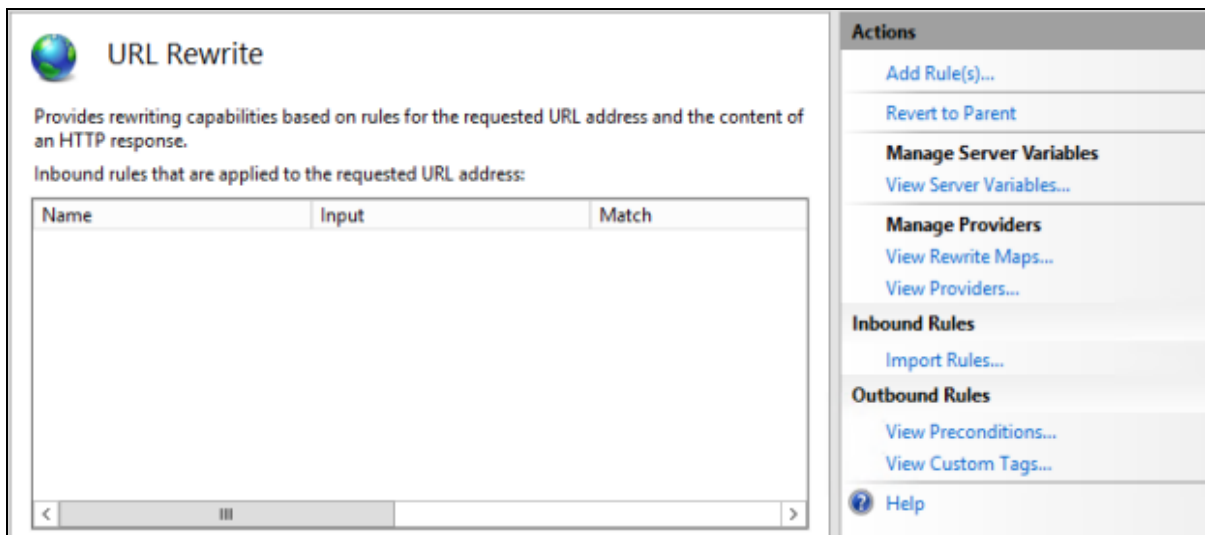


Figure 3.49: Choosing to add a new rule

- In Figure 3.50 that then appears, click the **Blank Rule** option.

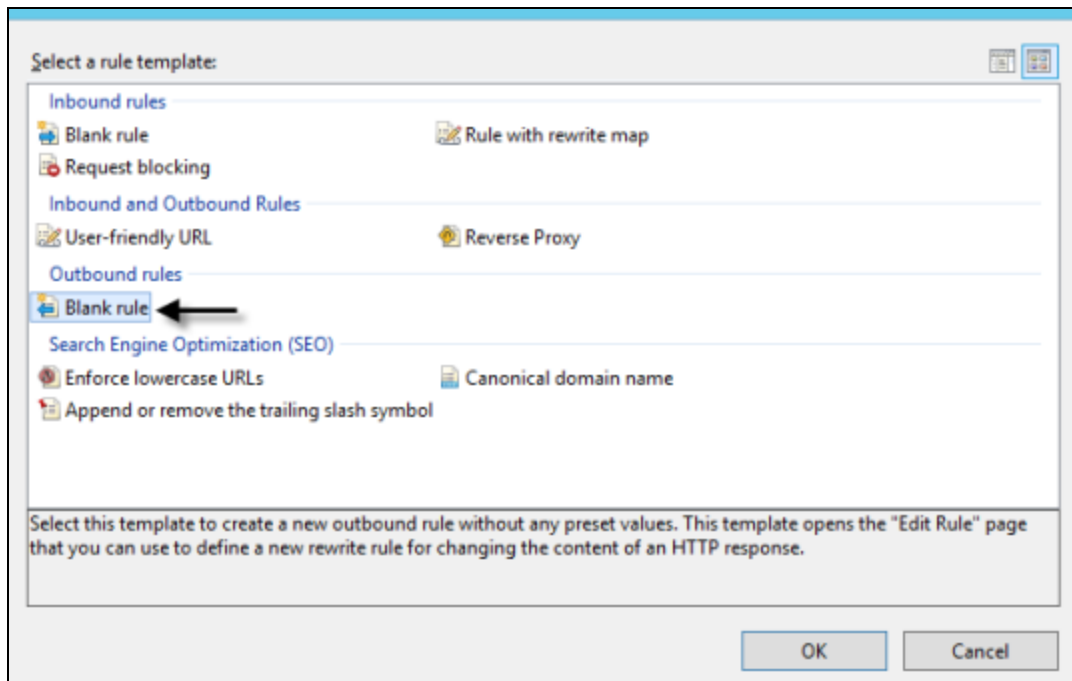
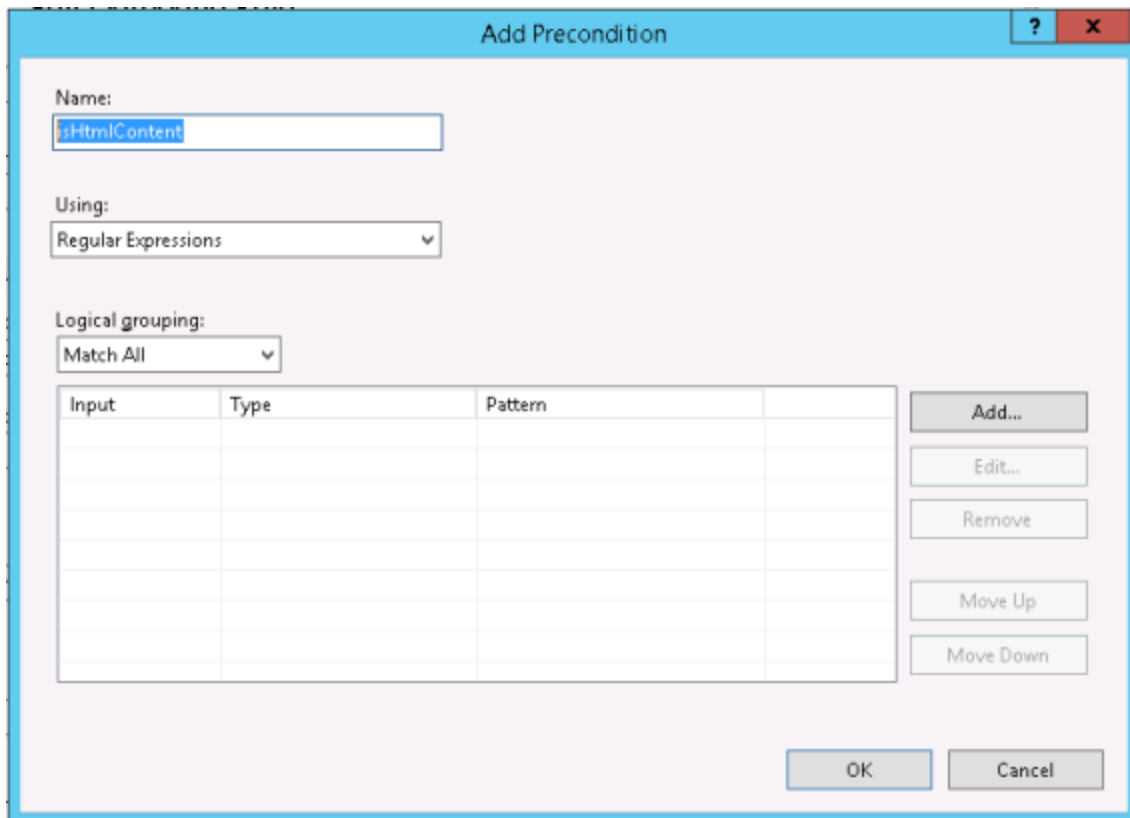


Figure 3.50: Selecting the Blank Rule option

5. Then, click the **OK** button in Figure 3.50.
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 3.51). 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.

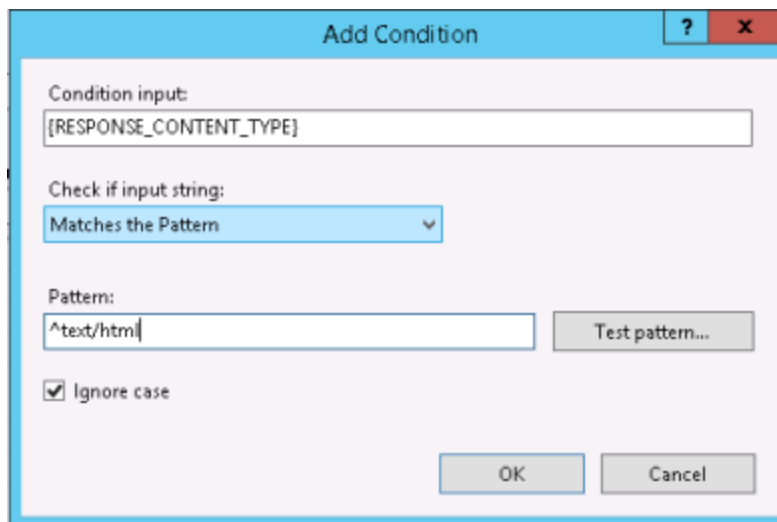


The "Add Precondition" dialog box is shown. It has a title bar with a question mark and a close button. The main area contains the following fields and controls:

- Name:** A text box containing "HtmlContent".
- Using:** A dropdown menu set to "Regular Expressions".
- Logical grouping:** A dropdown menu set to "Match All".
- Table:** A table with 4 columns: "Input", "Type", "Pattern", and an empty column. It is currently empty.
- Buttons:** "Add...", "Edit...", "Remove", "Move Up", and "Move Down" are located to the right of the table. "OK" and "Cancel" are at the bottom right.

Figure 3.51: Defining the properties of the new precondition

8. When Figure 3.52 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 3.52.



The "Add Condition" dialog box is shown. It has a title bar with a question mark and a close button. The main area contains the following fields and controls:

- Condition input:** A text box containing "{RESPONSE_CONTENT_TYPE}".
- Check if input string:** A dropdown menu set to "Matches the Pattern".
- Pattern:** A text box containing "^text/html".
- Test pattern...:** A button next to the pattern text box.
- Ignore case:** A checked checkbox.
- Buttons:** "OK" and "Cancel" are at the bottom right.

Figure 3.52: Providing the condition input, input string, and pattern

9. You will now return to Figure 3.53. Now, in the **Pattern** text box therein, type `</head[^>]*>`, as indicated by Figure 3.53.

Figure 3.53: Entering the pattern for the rule

10. Then, scroll down Figure 3.54 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 charset='UTF-8' type='text/javascript'> window['egrum-start_time'] = new Date
().getTime(); </script>

#2 :

<script charset='UTF-8' type='text/javascript'> window['Site_Name'] = 'BE_Site';
window['beacon-url'] = 'http://xyz.com';</script>

<script src='http://xyz.com/rumcollector/egrum.js' async>
```

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 3.54. Make sure that you append the line with the entry {R:0}. So, your complete Action Properties specification will be as follows:

```
<script charset='UTF-8' type='text/javascript'> window['egrum-start_time'] = new Date().getTime(); </script>{R:0}
```

Figure 3.54: Specifying the line of code to be auto-injected in the head

11. Finally, click the **Apply** button in Figure 3.55.

Figure 3.55: 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 3.56).

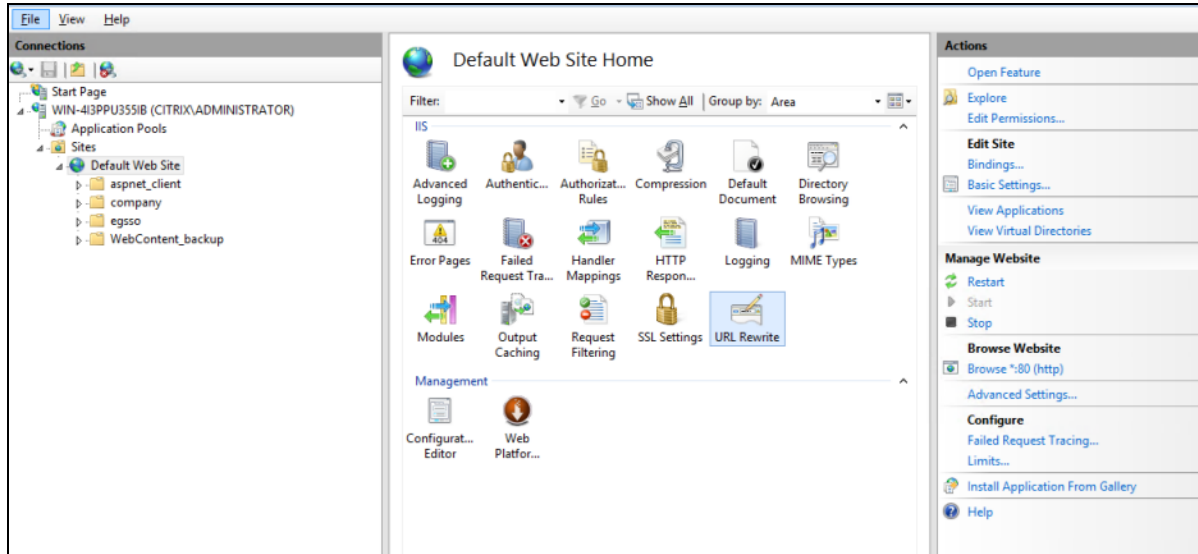


Figure 3.56: Accessing the URL rewrite option

13. When Figure 3.57 appears, select the **Add Rule(s)** option.

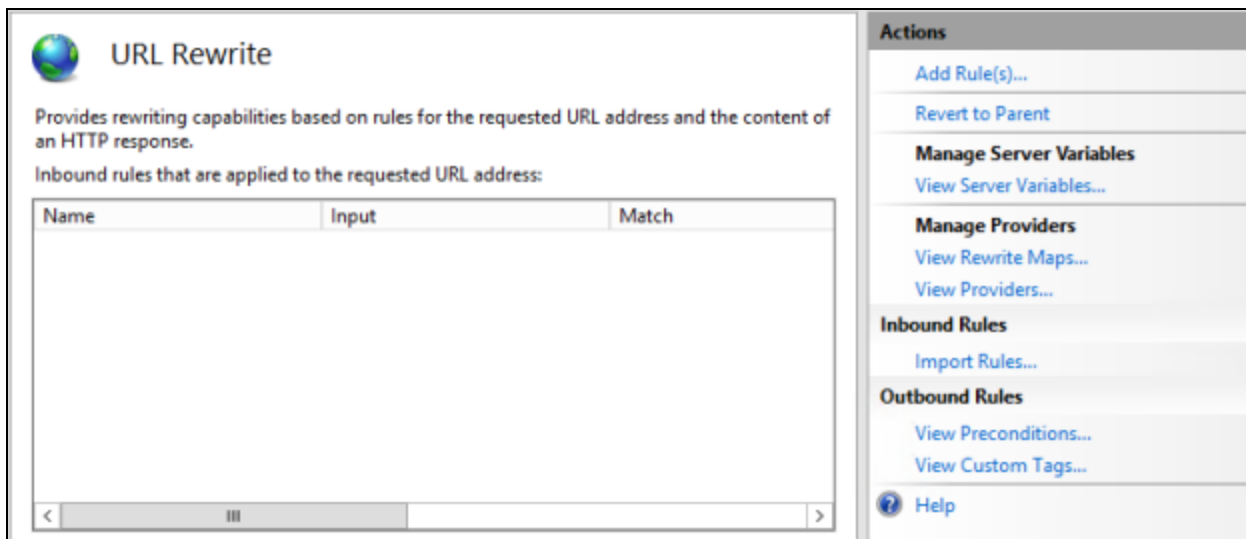


Figure 3.57: Choosing to add a new rule

14. In Figure 3.58 that then appears, click the **Blank Rule** option.

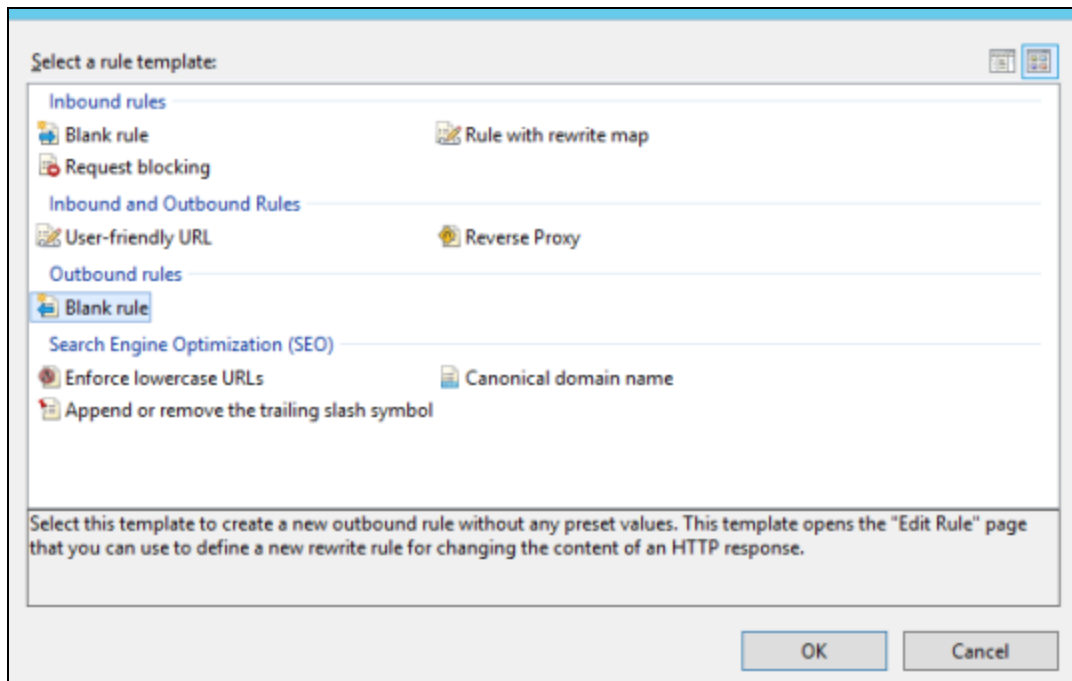


Figure 3.58: Selecting the Blank Rule option

15. Then, click the **OK** button in Figure 3.58.
16. In Figure 3.59 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[^>]*>`.

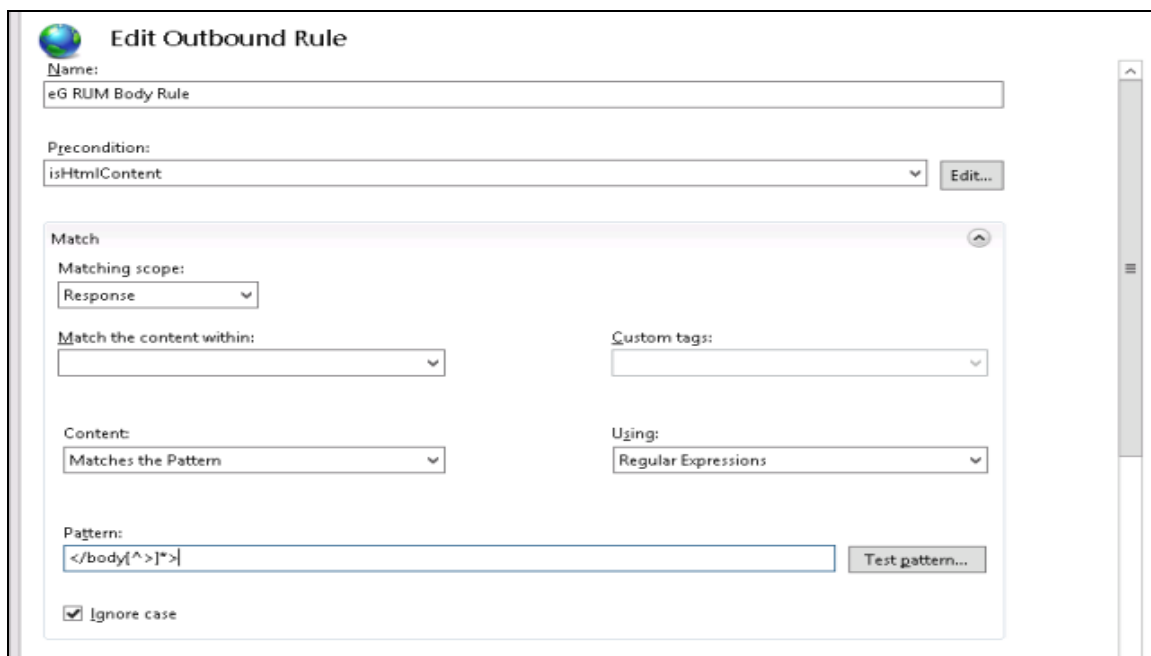


Figure 3.59: Adding the eG RUM Body Rule

17. Then, scroll down Figure 3.59 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 egrum.js script should be injected into the body. For instance, consider the sample JavaScript code snippet below:

```

<!-- RUM Header -->

#1:

<script charset='UTF-8' type='text/javascript'> window['egrum-start_time'] = new Date
().getTime(); </script>

#2 :

<script charset='UTF-8' type='text/javascript'> window['Site_Name'] = 'BE_Site';
window['beacon-url'] = 'http://xyz.com';</script>
<script src='http://xyz.com/rumcollector/egrum.js' async> </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 3.60. Make sure that you append the line with the entry `{R:0}`. So, your complete **Action Properties** specification will be as follows:

```

<script charset='UTF-8' type='text/javascript'> window['Site_Name'] = 'BE_Site';

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

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

```


![Screenshot of the 'Edit Outbound Rule' dialog box. The 'Content' dropdown is set to 'Matches the Pattern'. The 'Using' dropdown is set to 'Regular Expressions'. The 'Pattern' text box contains the regular expression '</body[^\>]*>'. There is a 'Test pattern...' button next to it. The 'Ignore case' checkbox is checked. The 'Conditions' section is empty. The 'Action' section shows 'Action type' set to 'Rewrite'. The 'Action Properties' section shows 'Value' set to 'window[\](\"http://xyz.com/rumcollector/egrum.js\")

Figure 3.60: Specifying the line of code to be auto-injected in the body

18. Finally, click the **Apply** button in Figure 3.61.

Figure 3.61: 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.

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 3.62.

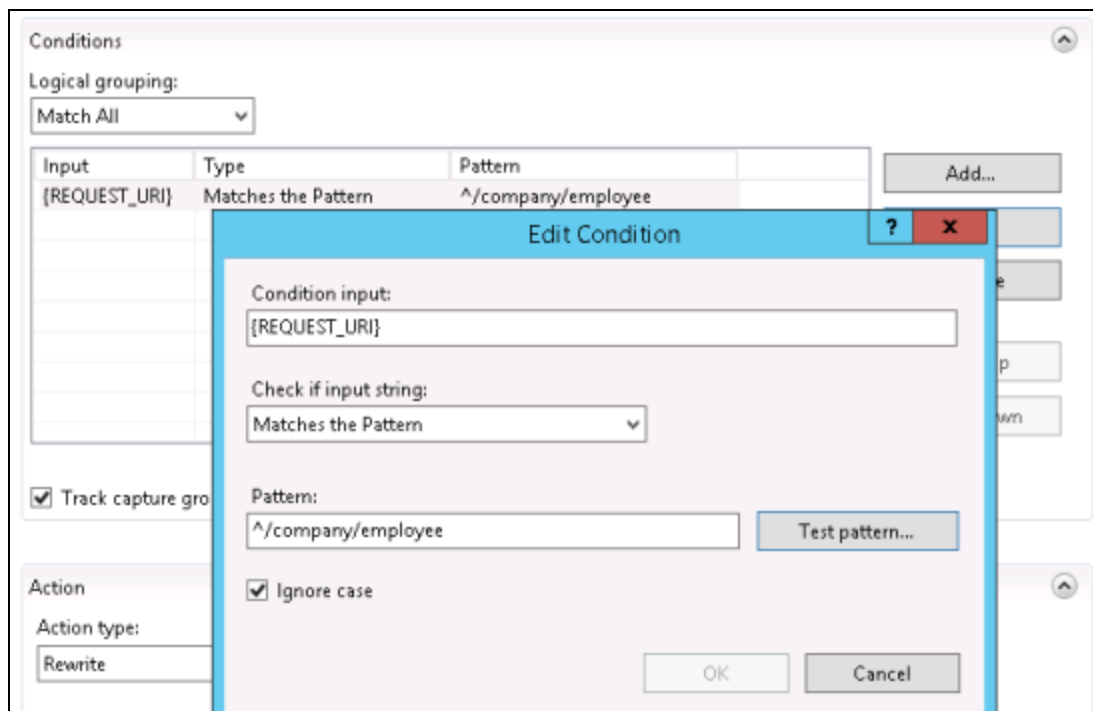


Figure 3.62: Auto-injecting the code snippet in web pages that match specific URL patterns

3.4.1.1 Verifying Whether a Web Application is a 32-bit Application or 64-bit Application

To achieve this, do the following:

1. Open the IIS Manager and expand the node representing the target IIS web server in the tree-structure in the left panel of the manager.
2. Click on the Application Pools sub-node in the tree-structure. The right panel will then change to display all the application pools. From this list, pick the application pool that contains the web application being monitored, and then select Advanced Settings from the list of options to its right.
3. In the General section of the Advanced Settings window that pops up, check the status of the Enable 32-Bit Applications flag. If this flag is True, it implies that the target web application is a 32-bit application. If this flag is False, then it means that the target web application is a 64-bit application. Figure 3.63 below clearly depicts steps 1-3 above.

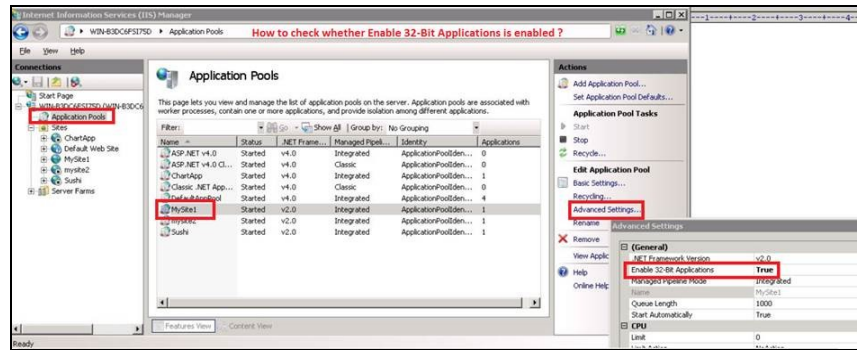


Figure 3.63: Verifying whether a web application is a 32-bit or a 64-bit application

3.4.1.2 Enabling the Failed Request Log for IIS

Request-based tracing is available both in stand-alone IIS Servers and on Azure web apps and provides a way to determine what exactly is happening with your requests and why, provided that you can reproduce the problem that you are experiencing. Problems like poor performance on some requests, or authentication-related failures on other requests, or the server 500 error from ASP or ASP.NET can often be difficult to troubleshoot--unless you have captured the trace of the problem when it occurs.

To be able to capture and troubleshoot issues related to injecting the RUM script through the URL Rewrite module of IIS, you should have request-based tracing enabled. To achieve this, follow the broad steps below:

- After finishing the default installation of IIS 7, install the tracing role service.
- Enable failed request tracing at the site level, application level, or directory level.
- Configure failure definitions.

Each of these steps are detailed below.

Installing the Tracing Role service

To install the tracing role service on Windows server 2012 / 2012 R2, follow the steps below:

1. On the taskbar, click **Server Manager**.
2. In **Server Manager**, click the **Manage** menu, and then click **Add Roles and Features**.
3. In the **Add Roles and Features** wizard, click **Next**. Select the installation type and click **Next**. Select the destination server and click **Next**.
4. On the **Server Roles** page, expand **Web Server (IIS)**, expand **Web Server**, expand **Health and Diagnostics**, and then select **Tracing**. Click **Next**.

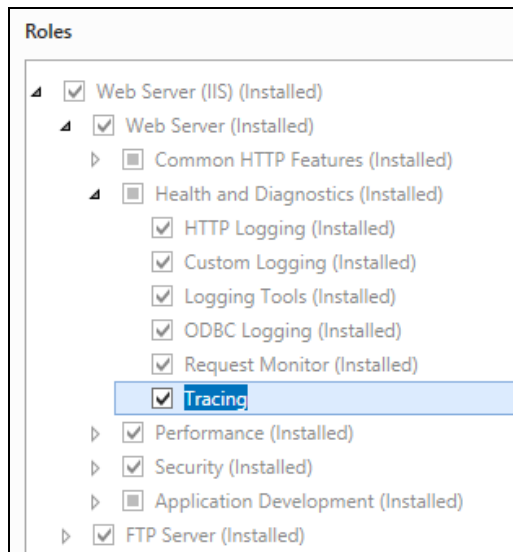


Figure 3.64: Installing the tracing role service on Windows 2012

5. On the **Select features** page, click **Next**.
6. On the **Confirm installation selections** page, click **Install**.
7. On the **Results** page, click **Close**.

To install the tracing role service on Windows 8 or 8.1, follow the steps below:

1. On the **Start** screen, move the pointer all the way to the lower left corner, right-click the **Start** button, and then click **Control Panel**.
2. In **Control Panel**, click **Programs and Features**, and then click **Turn Windows features on or off**.
3. Expand **Internet Information Services**, expand **World Wide Web Services**, expand **Health and Diagnostics**, and then select **Tracing**.

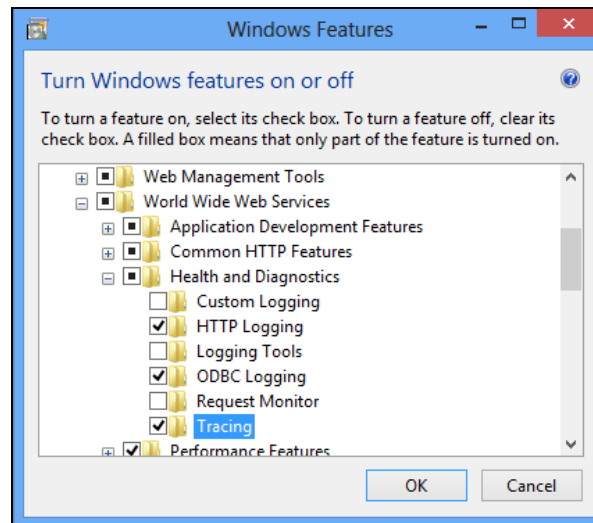


Figure 3.65: Installing the tracing role service on Windows 8 or 8.1

4. Click **OK** and then, click **Close**.

To install the tracing role service on Windows 2008 / 2008 R2, follow the steps below:

1. On the taskbar, click **Start**, point to **Administrative Tools**, and then click **Server Manager**.
2. In the **Server Manager** hierarchy pane, expand **Roles**, and then click **Web Server (IIS)**.
3. In the **Web Server (IIS)** pane, scroll to the **Role Services** section, and then click **Add Role Services**.
4. On the **Select Role Services** page of the **Add Role Services Wizard**, select **Tracing**, and then click **Next**.

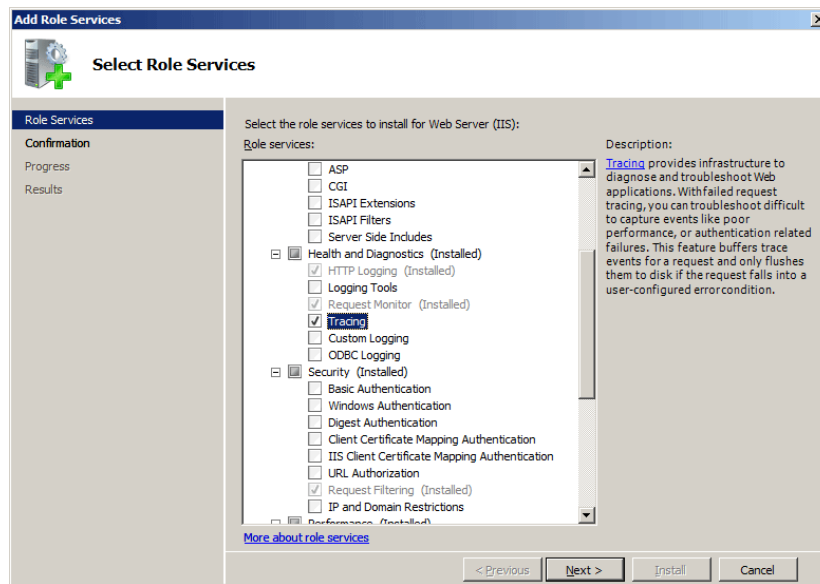


Figure 3.66: Installing the tracing role service on Windows 2008/2008 R2

5. On the **Confirm Installation Selections** page, click **Install**.
6. On the **Results** page, click **Close**.

To install the tracing role service on Windows Vista or 7, follow the steps below:

1. On the task bar, click **Start**, and then click **Control Panel**.
2. In **Control Panel**, click **Programs and Features**, and then click **Turn Windows features on or off**.
3. Expand **Internet Information Services**, expand **World Wide Web Services**, expand **Health and Diagnostics**, and then select **Tracing**.

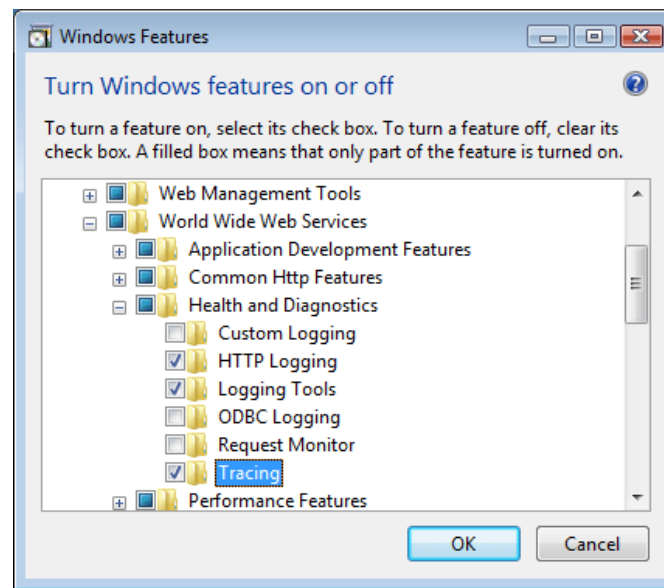


Figure 3.67: Installing the tracing role service on Windows 7 or Vista

4. Click **OK**.

Enabling tracing

1. Open **Internet Information Services (IIS) Manager**:
 - If you are using Windows Server 2012 or Windows Server 2012 R2:
 - On the taskbar, click **Server Manager**, click **Tools**, and then click **Internet Information Services (IIS) Manager**.
 - If you are using Windows 8 or Windows 8.1:
 - Hold down the Windows key, press the letter X, and then click **Control Panel**.
 - Click **Administrative Tools**, and then double-click **Internet Information Services (IIS) Manager**.
 - If you are using Windows Server 2008 or Windows Server 2008 R2:
 - On the taskbar, click **Start**, point to **Administrative Tools**, and then click **Internet Information Services (IIS) Manager**.
 - If you are using Windows Vista or Windows 7:
 - On the taskbar, click **Start**, and then click **Control Panel**.

- Double-click **Administrative Tools**, and then double-click **Internet Information Services (IIS) Manager**.
2. In the **Connections** pane, select the server connection, site, application, or directory for which you want to configure failed request tracing.
3. In the **Actions** pane, click **Failed Request Tracing...**

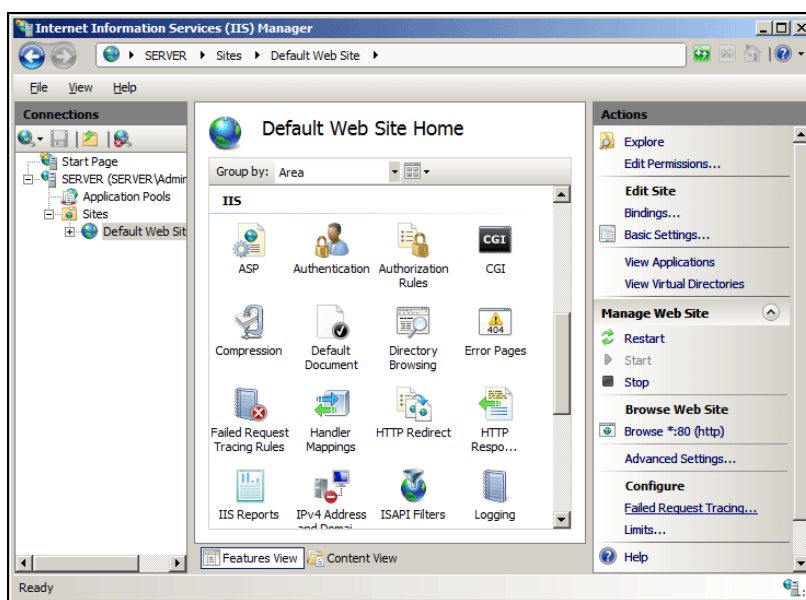


Figure 3.68: Clicking Failed Request Tracing

4. In the **Edit Web Site Failed Request Tracing Settings** dialog box, select the **Enable** check box to enable tracing, leave the default value or type a new directory where you want to store failed request log files in the **Directory** box, type the number of failed request trace files you want to store in the **Maximum number of trace files** box, and then click **OK**.

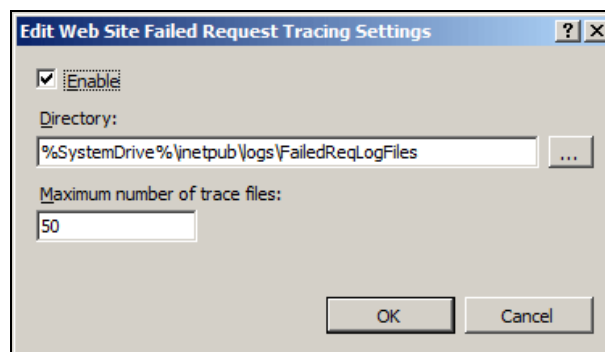


Figure 3.69: Enabling tracing

Configuring failure definitions

1. Open **Internet Information Services (IIS) Manager**:
 - If you are using Windows Server 2012 or Windows Server 2012 R2:
 - On the taskbar, click **Server Manager**, click **Tools**, and then click **Internet Information Services (IIS) Manager**.
 - If you are using Windows 8 or Windows 8.1:
 - Hold down the Windows key, press the letter X, and then click **Control Panel**.
 - Click **Administrative Tools**, and then double-click **Internet Information Services (IIS) Manager**.
 - If you are using Windows Server 2008 or Windows Server 2008 R2:
 - On the taskbar, click **Start**, point to **Administrative Tools**, and then click **Internet Information Services (IIS) Manager**.
 - If you are using Windows Vista or Windows 7:
 - On the taskbar, click **Start**, and then click **Control Panel**.
 - Double-click **Administrative Tools**, and then double-click **Internet Information Services (IIS) Manager**.
2. In the **Connections** pane, go to the connection, site, application, or directory for which you want to configure failed request tracing.
3. In the **Home** pane, double-click **Failed Request Tracing Rules**.

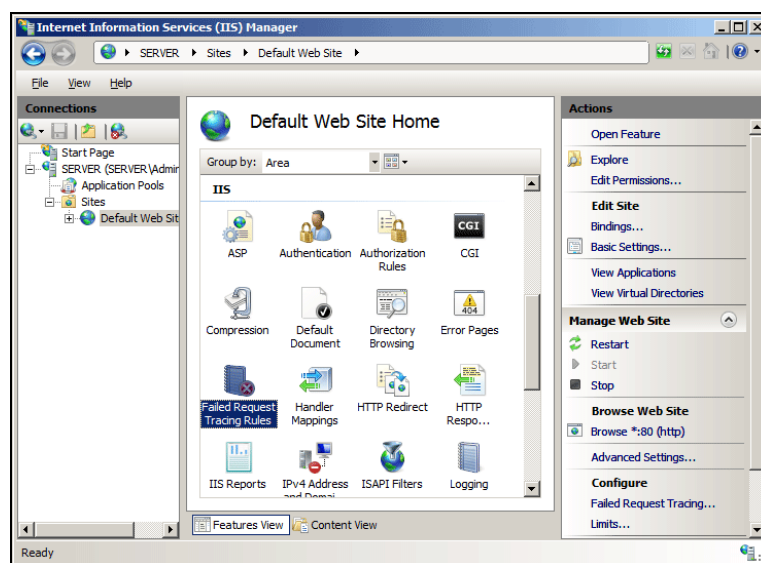


Figure 3.70: Clicking Failed Request Tracing Rules

4. In the **Actions** pane, click **Add...**
5. On the **Specify Content to Trace** page of the **Add Failed Request Tracing Rule Wizard**, select the content type you want to trace, and then click **Next**.

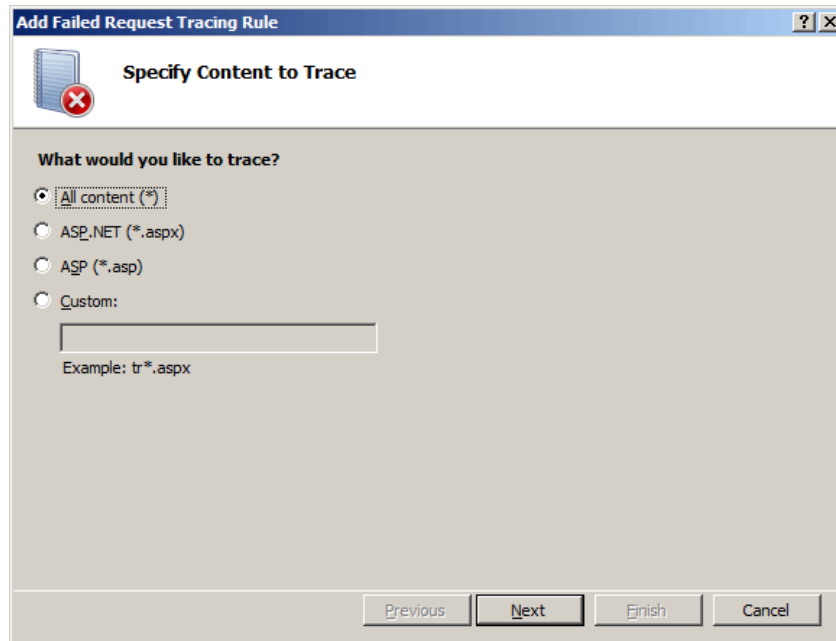
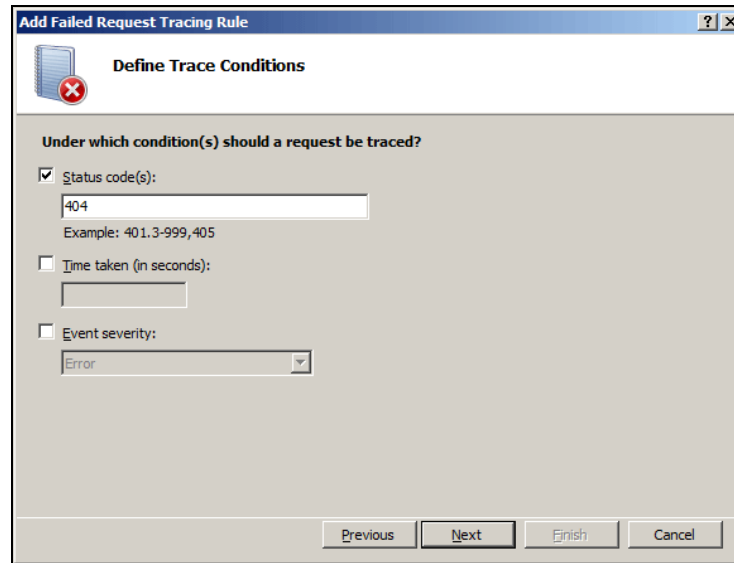


Figure 3.71: Selecting the content type to trace

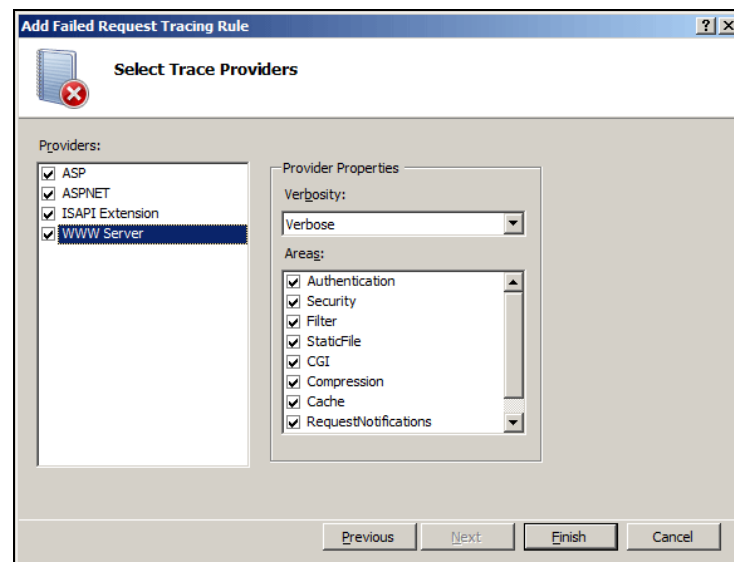
6. On the **Define Trace Conditions** page, select the conditions you want to trace, and then click **Next**. Trace conditions can include any combination of status codes, a time limit that a request should take, or the event severity. If you specify all conditions, the first condition that is met generates the failed request trace log file.



The screenshot shows the 'Add Failed Request Tracing Rule' dialog box, specifically the 'Define Trace Conditions' tab. The dialog has a title bar with a question mark and a close button. Below the title bar is a red 'X' icon and the text 'Define Trace Conditions'. The main area is titled 'Under which condition(s) should a request be traced?'. There are three conditions listed: 'Status code(s):' with a checked checkbox and a text box containing '404' (with an example '401,3-999,405' below it); 'Time taken (in seconds):' with an unchecked checkbox and an empty text box; and 'Event severity:' with an unchecked checkbox and a dropdown menu set to 'Error'. At the bottom are four buttons: 'Previous', 'Next', 'Finish', and 'Cancel'.

Figure 3.72: Defining trace conditions

7. On the **Select Trace Providers** page, select one or more of the trace providers under **Providers**.



The screenshot shows the 'Add Failed Request Tracing Rule' dialog box, specifically the 'Select Trace Providers' tab. The dialog has a title bar with a question mark and a close button. Below the title bar is a red 'X' icon and the text 'Select Trace Providers'. The main area is divided into two sections. On the left, under the heading 'Providers:', there is a list box with four items: 'ASP', 'ASP.NET', 'ISAPI Extension', and 'WWW Server', all of which are checked. On the right, under the heading 'Provider Properties', there is a 'Verbosity:' dropdown menu set to 'Verbose' and a list box for 'Areag:' with seven items: 'Authentication', 'Security', 'Filter', 'StaticFile', 'CGI', 'Compression', and 'Cache', all of which are checked. At the bottom are four buttons: 'Previous', 'Next', 'Finish', and 'Cancel'.

Figure 3.73: Selecting one/more trace providers

8. On the **Select Trace Providers** page, select one or more of the verbosity levels under **Verbosity**.

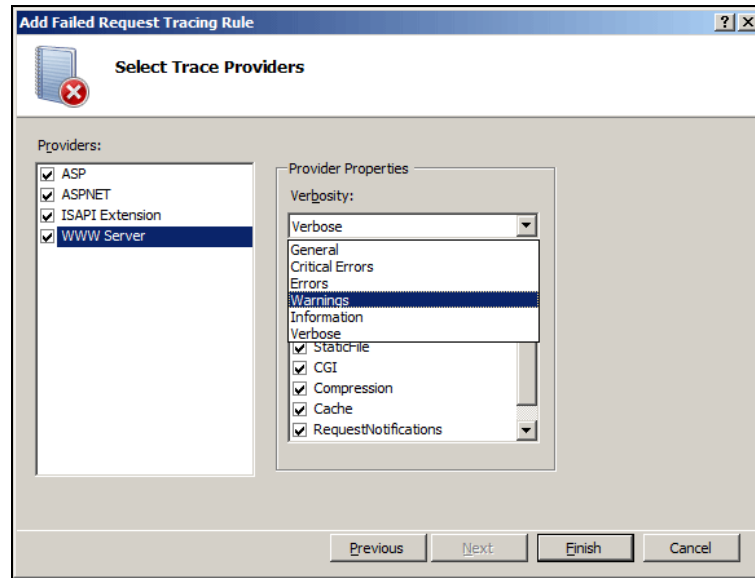


Figure 3.74: Selecting one/more verbosity levels

9. If you selected the ASPNET or WWW Server trace provider in step 8, select one or more functional areas for the provider to trace under **Areas of the Select Trace Providers** page.
10. Click **Finish**.

3.4.2 Instrumenting an Apache Web Server Using Container-Assisted Injection

If you are using the Apache web server 2.2.7 (and above), you can use the container substitution pattern to automatically inject the eG Real User Monitoring JavaScript agent into your web pages. This allows you to enable the RUM functionality for your web pages without needing to make any code changes.

Note:

The table below lists the command to be run on each operating system to determine the version of the Apache server running on that system.

Operating System of Apache Server	Command for Determining Apache Version
Debian and Mac OS	apachectl -v
Red Hat Linux and Amazon's EC2 Linux	httpd -v or /usr/sbin/httpd -v
Other versions of Linux	apache2 -v or /usr/sbin/apache2 -v If the aforesaid commands do not work, then it could be because the

	login user does not have permissions to execute the commands. In this case, use the sudo command to grant the required permissions to the login user and run either of the commands again.
Windows	"C:\Program Files\[Apache directories]\bin\httpd.exe" -v

This mechanism intercepts the response object as it is being returned and makes a string substitution containing the required JavaScript tag at the appropriate place in the HTML.

Note:

Container-assisted RUM script injection can be used to RUM-enable only those web servers that support the *mod_substitute* and *mod_filter* modules. If a web server does not support these modules or supports only one of them, then such web servers cannot be RUM-enabled using the container-based injection method.

The IBM HTTP server 8.x (and below) does not include the *mod_substitute* module. This means that the container-based injection method cannot be used to RUM-enable this server. You will hence have to use one of the alternate approaches depicted by Figure 1 to RUM-enable the IBM HTTP server 8.x (and below). In the case of the IBM HTTP server 9.x (and above) on the other hand, both the *mod_substitute* and *mod_filter* modules are supported. The container-based injection method can hence be used here. Use the procedure detailed in the Instrumenting an IBM HTTP Server v9.x (and above) Using Container-Assisted Injection topic to RUM-enable an IBM HTTP Server 9.x (and above).

The instrumentation steps are as follows:

1. First, add the web site (on the Apache web server) to be monitored as a *Real User Monitor* component, using the eG admin interface. When adding, you will find that the RUM code snippet that you need to use for instrumenting the Apache web server is displayed in the eG admin interface. Copy the code snippet.
2. Open the global Apache **httpd.conf** file and search for the following commands:


```
LoadModule substitute_module modules/mod_substitute.so
LoadModule filter_module modules/mod_filter.so
```
3. Once you find the commands, check if they are commented. If they are, then uncomment them. On the other hand, if they are already uncommented, leave them as is.
4. Create a directory called **egrum** on the system hosting the Apache server.

5. Create a file named **egrum.conf** in that directory. Enter the following in that file:

```
<Location />

AddOutputFilterByType SUBSTITUTE text/html

substitute "s|</head>|
```

6. Then, after the "s|</head>| tag in the above specification, paste the code snippet you copied from the eG admin interface in step 1 of this procedure. Make sure that you paste the snippet **as a single line**. The contents of the egrum.conf file will now look as shown below:

```
<Location />

AddOutputFilterByType SUBSTITUTE text/html

substitute "s|</head>|<script charset='UTF-8' type='text/javascript'> window['egrum-
start_time']=new Date().getTime();window['Site_Name']='SITENAME';window['beacon-
url']='http://RUM_COLLECTOR_DOMAIN_NAME:PORT';</script><script src='http://RUM_
COLLECTOR_DOMAIN_NAME:PORT/rumcollector/egrum.js' async> </script>
```

After inserting the code snippet, take care to replace the area highlighted in **Green**, in the sample snippet above, with the **exact domain name and port number of the RUM collector**.

7. Then, after the **</script>** tag highlighted in Red in the sample snippet above, type the following:

```
</head>|inq"

</Location>
```

8. The complete contents of the egrum.conf file should look as shown below:

```
<Location />

AddOutputFilterByType SUBSTITUTE text/html

substitute "s|</head>|<script charset='UTF-8' type='text/javascript'> window['egrum-
start_time']=new Date().getTime();window['Site_Name']='SITENAME';window['beacon-
url']='http://RUM_COLLECTOR_DOMAIN_NAME:PORT';</script><script src='http://RUM_
COLLECTOR_DOMAIN_NAME:PORT/rumcollector/egrum.js' async> </script> </head>|inq"

</Location>
```

In the sample contents above, the **</head>** tag highlighted in Red indicates that this code snippet looks for an exact **</head>** tag match. Alternatively, you can alter the code snippet to look for a **<body>** tag match, if required.

9. Then, save the **egrum.conf** file.

10. Next, add the location of the **egrum.conf** file to **httpd.conf**, as indicated below:

```
Include [absolutePathTo]/egrum.conf
```

For instance, if the **egrum.conf** file resides in the C:\egrum folder, then the above specification will be:

```
Include C:\egrum\egrum.conf
```

11. Finally, save the file and restart the Apache web server.

Note:

If Apache HTTP server is configured as a reverse proxy, you must use the *Location* directive based method, with the **ProxyPass** and **ProxyPassReverse** directives, as indicated in the code block below:

```
<Location />
```

```
ProxyPass http://mywebsite.index.net/
```

```
ProxyPassReverse http://mywebsite.index.net/
```

```
AddOutputFilterByType SUBSTITUTE text/html
```

```
substitute "s|</head>|<script charset='UTF-8' type='text/javascript'> window['egrum-  
start_time'] = new Date().getTime();window['Site_Name'] = 'SITENAME';window['beacon-  
url'] = 'http://RUM_COLLECTOR_DOMAIN_NAME:PORT';</script><script src='http://RUM_  
COLLECTOR_DOMAIN_NAME:PORT/rumcollector/egrum.js' async> </script> </head>|inq"
```

```
</Location>
```

Troubleshooting Tips:

The table below lists the command that you need to run on each operating system to check for syntax errors in changes you made in the configuration files (**egrum.conf** and **httpd.conf**). If no syntax errors are detected, then each of these commands will output a message to that effect.

Operating System of Apache Server	Command for checking whether any syntax errors exist in the configuration files
Debian and Mac OS	apachectl configtest
Red Hat Linux and Amazon's EC2 Linux	httpd -t or /usr/sbin/httpd -t

Other versions of Linux	<pre>apache2 -t or /usr/sbin/apache2 -t</pre> If the aforesaid commands do not work, then it could be because the login user does not have permissions to execute the commands. In this case, use the sudo command to grant the required permissions to the login user and then run either of the commands again.
Windows	<pre>"C:\Program Files\[Apache directories]\bin\httpd.exe" -t</pre>

3.4.3 Instrumenting an NGINX Server Using Container-Assisted Injection

If you are using an NGINX server as a reverse proxy, you can use the container substitution pattern to automatically inject the eG RUM JavaScript into your web pages. This allows you to enable the RUM functionality for your web pages without needing to make any code changes.

This mechanism intercepts the response object as it is being returned and makes a string substitution containing the required JavaScript at the appropriate place in the HTML response.

To RUM-enable a web site / web application on an NGINX web server, follow the steps below:

1. The substitution mechanism requires the *ngx_http_sub_module* for RUM-enabling a web site / web application on NGINX. This module is an extension and hence, may not be included in an NGINX build by default. Therefore, first check whether/not the NGINX build in use is built with this module. For that, run the following command on the NGINX web server host:

```
nginx -V
```

2. If the output returned by this command does not include the *ngx_http_sub_module*, then it indicates that the NGINX web server in use has not been configured with the *ngx_http_sub_module* module. In this case, make sure that the NGINX build is first enabled with the **with-http_sub_module** configuration parameter, and then proceed to RUM-enable it.
3. Next, edit the global **nginx.conf** file. Modify the location context to replace the <head> tag of the file with the <head> tag and the JavaScript agent scripts, as indicated below:


```

server {
    listen 80;
    server_name localhost;

    #charset koi8-r;

    #access_log logs/host.access.log main;

    location / {
        root html;
        index index.html index.htm;

        sub_filter <head> '<head><script charset="UTF-8" type="text/javascript"> window
["egrum-start_time"]=new Date().getTime();window["Site_Name"]=
"SITENAME";window["beacon-url"]
="http://RUM_COLLECTOR_DOMAIN_NAME:PORT";</script><script
src="http://RUM_COLLECTOR_DOMAIN_NAME:PORT/rumcollector/egrum.js" async>
</script>';
        sub_filter_once on;
    }
}

```

Figure 3.75: Editing the nginx.conf file

When replacing, ensure the following:

- The `<head>` tag should begin with a single quote, as depicted by Figure 1.
 - All single quotes within the head tag should be replaced by double quotes as indicated by Figure 1.
 - The code block highlighted in Figure 1 is the one that you will have to insert in the **nginx.conf** file. Make sure that you replace the variable **RUM_COLLECTOR_DOMAIN_NAME:PORT** in the code block, with the fully qualified domain name of the RUM collector and its port. Likewise, replace the **SITENAME** variable with the name of the RUM component that is being managed. The rule represented by this code block looks for an exact `<head>` tag match. Optionally, you can use the `<body>` tag or `<title>` tag too .
4. Finally, save the file and restart the NGINX server.
 5. Once this is done, then the JavaScript in the code block of Figure 3.75 will get automatically inserted into each web page of the web site/web application being managed, track the responsiveness of every web page, and return metrics to the eG agent.

3.4.4 Instrumenting an Oracle HTTP Server Using Container-based Injection

If you are using the Oracle HTTP Server 11G, you can use the container substitution pattern to automatically inject the eG Real User Monitoring JavaScript agent into your web pages. This allows you to enable the RUM functionality for your web pages without needing to make any code changes. This mechanism intercepts the response object as it is being returned and makes a string substitution containing the required JavaScript tag at the appropriate place in the HTML.

Note:

The container-based injection method (detailed in this topic) can be used to RUM-enable an Oracle HTTP server v11 (or above) only. For lower versions of the Oracle HTTP server (eg., 10G), you have to use any of the alternate instrumentation mechanisms depicted by Figure 1.

The instrumentation steps are as follows:

1. Open the global **httpd.conf** file and search for the following commands:

```
LoadModule substitute_module modules/mod_substitute.so
```

```
LoadModule filter_module modules/mod_filter.so
```

2. Once you find the commands, uncomment them.
3. Create a directory called **egrum** on the system hosting the Apache server.
4. Create a file named **egrum.conf** in that directory. The contents of the file should be similar to the snippet below:

```
<Location />

AddOutputFilterByType SUBSTITUTE text/html

substitute "s|</head>|<script charset='UTF-8' type='text/javascript'> window['egrum-
start_time'] = new Date().getTime();window['Site_Name'] = 'SITENAME';window['beacon-
url'] = 'http://RUM_COLLECTOR_DOMAIN_NAME:PORT';</script><script src='http://RUM_
COLLECTOR_DOMAIN_NAME:PORT/rumcollector/egrum.js' async> </script> </head>|inq"

</Location>
```

In the sample contents above, the **</head>** tag highlighted in Red indicates that this code snippet looks for an exact **</head>** tag match. Alternatively, you can alter the code snippet to look for a **<body>** tag match, if required.

Also, when inserting the code snippet, take care to replace the area highlighted in Green with the exact domain name and port number of the RUM collector.

5. Next, add the location of the **egrum.conf** file to **httpd.conf**, as indicated below:

```
Include [absolutePathTo]/egrum.conf
```

For instance, if the **egrum.conf** file resides in the C:\egrum folder, then the above specification will be:

```
Include C:\egrum\egrum.conf
```

6. Finally, save the file and restart the Oracle HTTP server.

Note:

If the Oracle HTTP server is configured as a reverse proxy, you must use the *Location* directive based method, with the **ProxyPass** and **ProxyPassReverse** directives as indicated by the code block below:

```
<Location />

ProxyPass http://mywebsite.index.net/

ProxyPassReverse http://mywebsite.index.net/

AddOutputFilterByType SUBSTITUTE text/html

substitute "s|</head>|<script charset='UTF-8' type='text/javascript'> window['egrum-
start_time'] = new Date().getTime();window['Site_Name'] = 'SITENAME';window['beacon-
url'] = 'http://RUM_COLLECTOR_DOMAIN_NAME:PORT';</script><script src='http://RUM_
COLLECTOR_DOMAIN_NAME:PORT/rumcollector/egrum.js' async> </script> </head>|inq"

</Location>
```

3.4.5 Instrumenting a Java-based Web Site / Web Application

As already mentioned, you can use the **egrum.jar** file that eG Enterprise provides to instrument a Java-based web site/web application that uses server side templates to auto-generate the common section. 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 3.43.
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 template that auto-generates the common section, in an editor. In the header of the template, insert the code that imports the helper class – *com.eg.RUMHandler* – into the template, as shown below:

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

<html>
```

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

4. 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>
```

5. Save the edited template and restart the web site/web application being monitored.
6. Now, test your web application locally and view the page source in your browser. The page markup should include `<!-- RUM Header -->`.
7. Repeat steps 3 to 7 above for each server side template that dynamically generates the common section.

Once this is done, then the JavaScript will be auto-injected into each web page that uses the common section, track requests to each of the web pages, and report response time metrics it collects to the RUM collector.

3.4.6 Instrumenting a DotNet-based Web Site / Web Application

As already mentioned, you can use the **Eg_RUM.dll** file that eG Enterprise provides to instrument a DotNet-based web site/web application uses server side templates to auto-generate the common section. 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 3.43.
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.

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 JavaScript will be auto-injected into each web page that uses the **Shared_Layouts.html** or **Default.master** files (as the case may be), track requests to each of the web pages, and report response time metrics it collects to the RUM collector.

3.4.7 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.

Component Information

Category	All
Component type	Real User Monitor
Nick name	RUM
RUM collector	Default Collector
Remote agent	192.168.8.26
Capture UserName	☐ Yes ☒ No

To instrument your html pages use the eG javascript agent

Download this if you have a Java web application that emits HTML to the front-end. Examples: JSF, JSP, Groovy:	egrum.jar	Download
Download this if you have a .NET web application such as ASP.NET MVC or WebForms:	Eg.RUM.dll	Download
Download this if you can edit the source code of your web pages. Examples: Plain HTML, JSP with embedded HTML, PHP with embedded HTML:	egrum.js	Download
Download this if want to embed this egrum.js into packaged applications such as SAP Fiori:	Code snippet + egrum.js	Download

Select the source file type on which eG RUM JavaScript code snippet to be included

☒ HTML ☐ Javascript

Include this line in your pages (html/jsp/aspx) immediately after the head tag

```
<!-- RUM Header -->
<script charset="UTF-8" type="text/javascript"> window['egrum-start_time'] = new
Date().getTime();
window['Site_Name'] = 'e75e3881-6e1e-4c47-9e4c-8508ae5ed2c2-1574317681316';
window['beacon-url'] = 'https://192.168.8.26:7077';
</script>
<script src='https://192.168.8.26:7077/rumcollector/egrum.js' async> </script>
```

Figure 3.76: 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 3.76 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*.

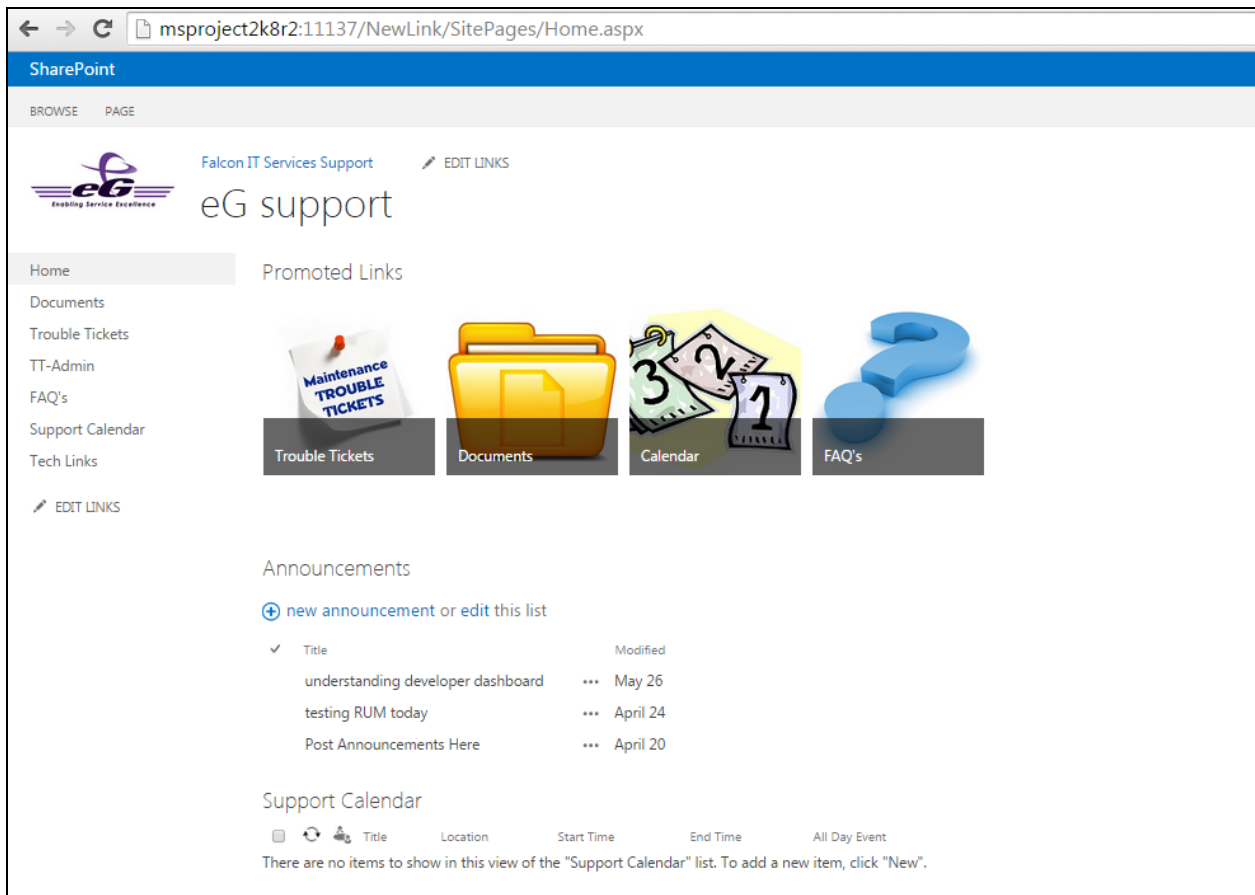


Figure 3.77: The Home page of the Sharepoint site to be monitored

5. Then, click on the **Settings** icon (indicated by Figure 3.78) at the right, top corner of the Home page to invoke a menu. From this menu, select the **Site settings** option.

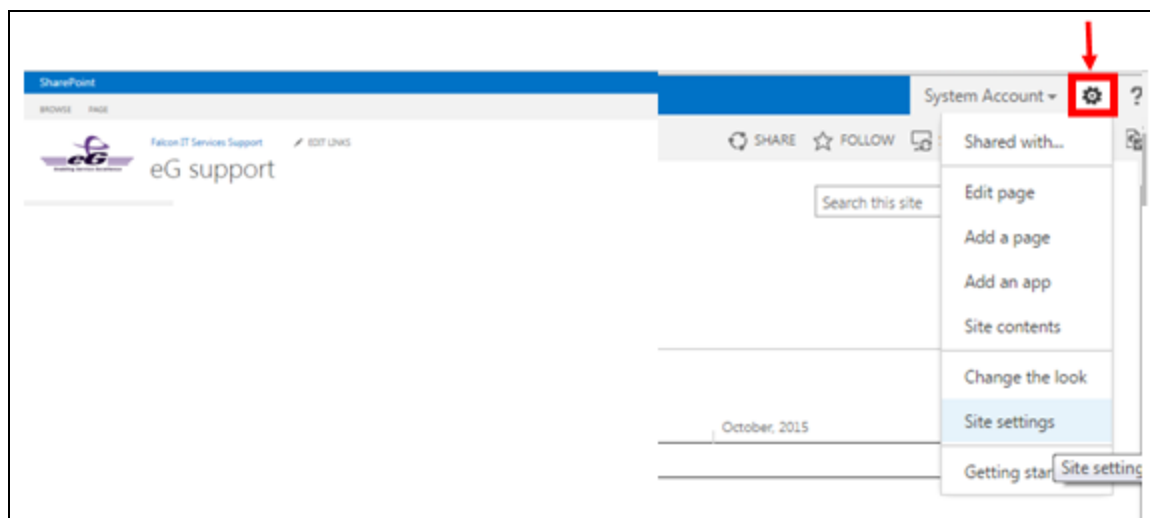


Figure 3.78: Selecting the Site settings option from the Home page

6. From the **Site Settings** window (see Figure 3.79) that then appears, select the **Master Pages** option in the **Web Designer Galleries** section.

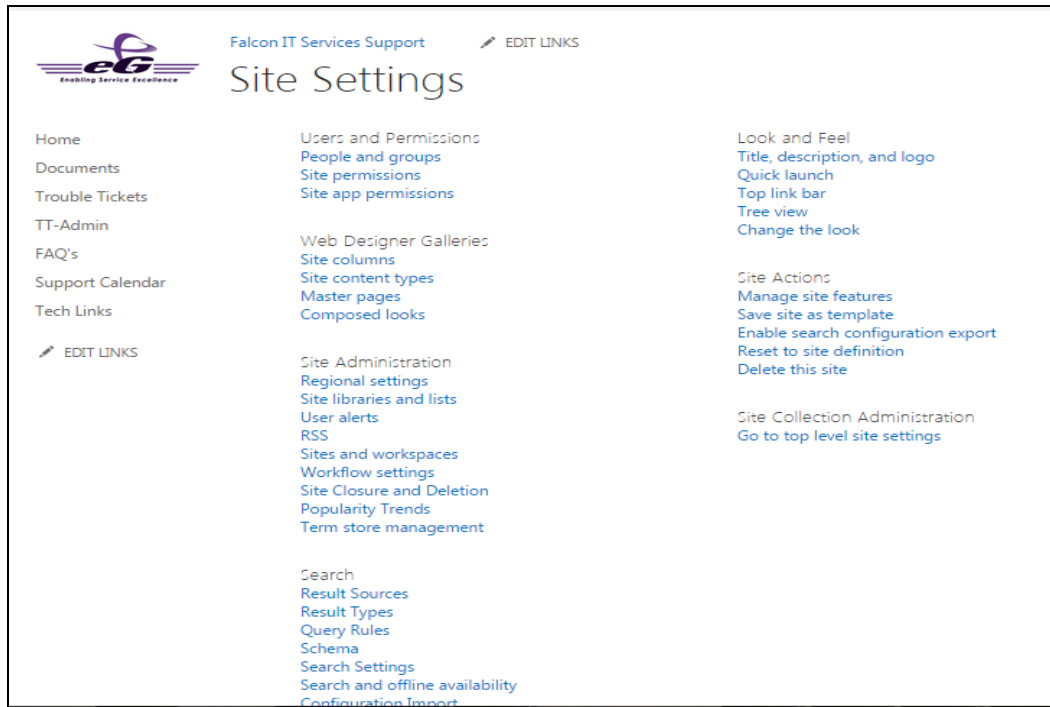
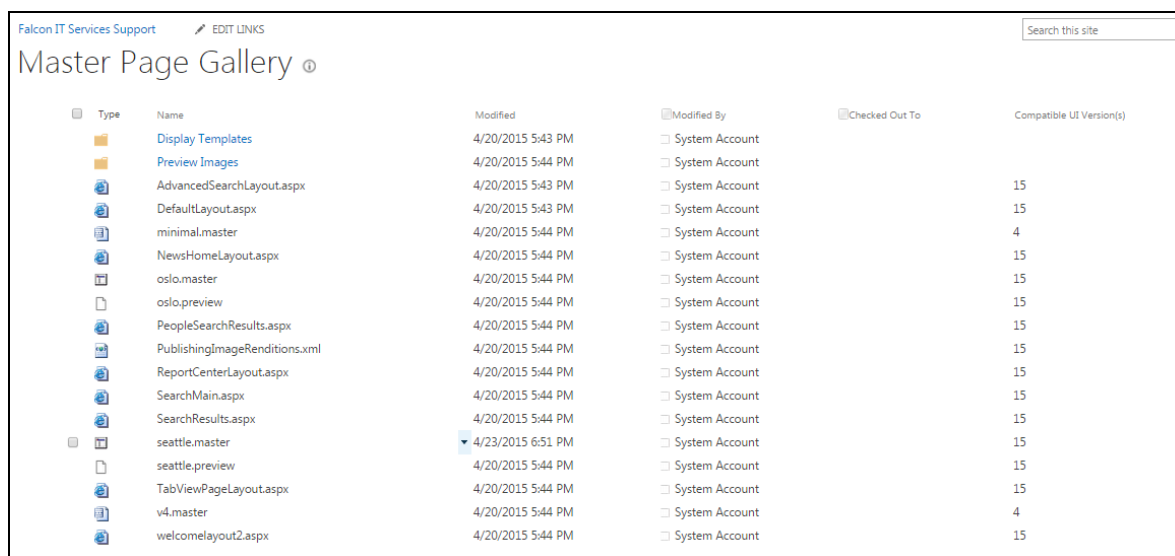


Figure 3.79: Selecting the Master pages option

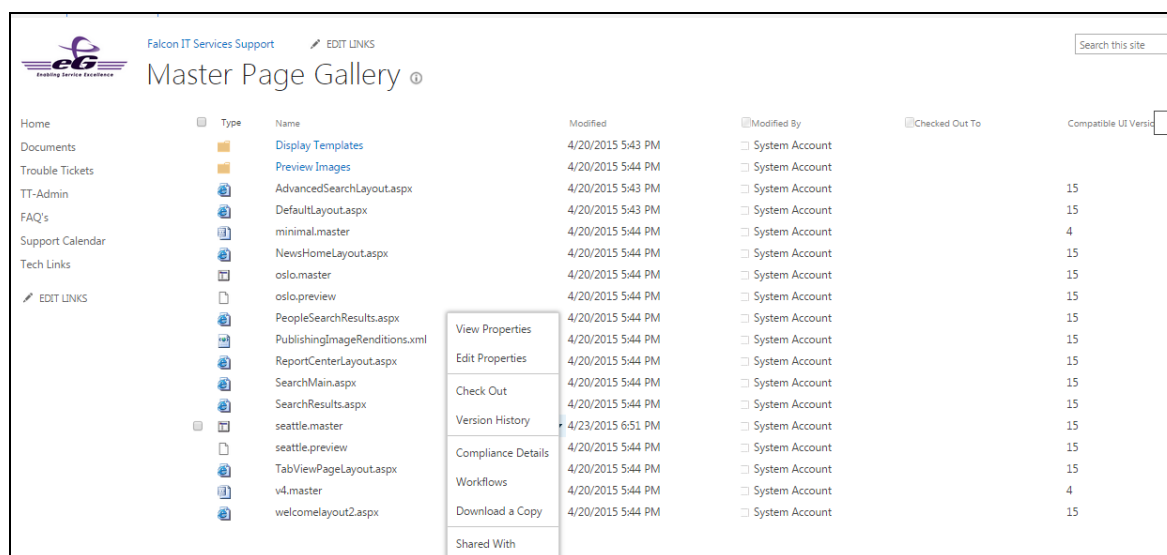
7. When the **Master Page Gallery** opens (see Figure 3.80), look for the **seattle.master** file (in the case of Sharepoint 2013) or the **v4.master** file (in the case of Sharepoint 2010), in the gallery. Once you find it, hover your mouse over the file name in the gallery. A down-arrow icon (as indicated by Figure 3.80), will then become visible alongside. Click on the down-arrow icon.



Type	Name	Modified	Modified By	Checked Out To	Compatible UI Version(s)
Folder	Display Templates	4/20/2015 5:43 PM	System Account		
Folder	Preview Images	4/20/2015 5:44 PM	System Account		
Page	AdvancedSearchLayout.aspx	4/20/2015 5:43 PM	System Account		15
Page	DefaultLayout.aspx	4/20/2015 5:43 PM	System Account		15
Page	minimal.master	4/20/2015 5:44 PM	System Account		4
Page	NewsHomeLayout.aspx	4/20/2015 5:44 PM	System Account		15
Page	oslo.master	4/20/2015 5:44 PM	System Account		15
Page	oslo.preview	4/20/2015 5:44 PM	System Account		15
Page	PeopleSearchResults.aspx	4/20/2015 5:44 PM	System Account		15
Page	PublishingImageRenditions.xml	4/20/2015 5:44 PM	System Account		15
Page	ReportCenterLayout.aspx	4/20/2015 5:44 PM	System Account		15
Page	SearchMain.aspx	4/20/2015 5:44 PM	System Account		15
Page	SearchResults.aspx	4/20/2015 5:44 PM	System Account		15
Page	seattle.master	4/23/2015 6:51 PM	System Account		15
Page	seattle.preview	4/20/2015 5:44 PM	System Account		15
Page	TabViewPageLayout.aspx	4/20/2015 5:44 PM	System Account		15
Page	v4.master	4/20/2015 5:44 PM	System Account		4
Page	welcomelayout2.aspx	4/20/2015 5:44 PM	System Account		15

Figure 3.80: Locating the seattle.master file on Sharepoint 2013

- Clicking on the down-arrow will invoke a menu. From this menu, select the **Check Out** option (see Figure 3.81).



Type	Name	Modified	Modified By	Checked Out To	Compatible UI Version(s)
Folder	Display Templates	4/20/2015 5:43 PM	System Account		
Folder	Preview Images	4/20/2015 5:44 PM	System Account		
Page	AdvancedSearchLayout.aspx	4/20/2015 5:43 PM	System Account		15
Page	DefaultLayout.aspx	4/20/2015 5:43 PM	System Account		15
Page	minimal.master	4/20/2015 5:44 PM	System Account		4
Page	NewsHomeLayout.aspx	4/20/2015 5:44 PM	System Account		15
Page	oslo.master	4/20/2015 5:44 PM	System Account		15
Page	oslo.preview	4/20/2015 5:44 PM	System Account		15
Page	PeopleSearchResults.aspx	4/20/2015 5:44 PM	System Account		15
Page	PublishingImageRenditions.xml	4/20/2015 5:44 PM	System Account		15
Page	ReportCenterLayout.aspx	4/20/2015 5:44 PM	System Account		15
Page	SearchMain.aspx	4/20/2015 5:44 PM	System Account		15
Page	SearchResults.aspx	4/20/2015 5:44 PM	System Account		15
Page	seattle.master	4/23/2015 6:51 PM	System Account		15
Page	seattle.preview	4/20/2015 5:44 PM	System Account		15
Page	TabViewPageLayout.aspx	4/20/2015 5:44 PM	System Account		15
Page	v4.master	4/20/2015 5:44 PM	System Account		4
Page	welcomelayout2.aspx	4/20/2015 5:44 PM	System Account		15

Figure 3.81: Checking out the master file

- Then, select the **Download a Copy** option from the same menu (see Figure 3.82) to download a copy of the master file to any location on the local host.

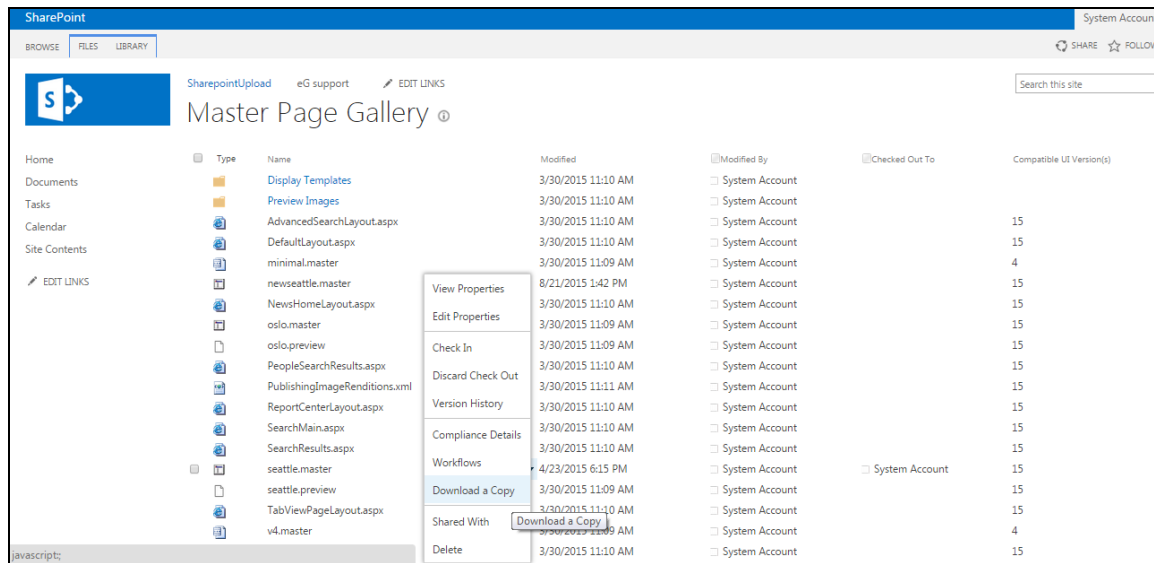
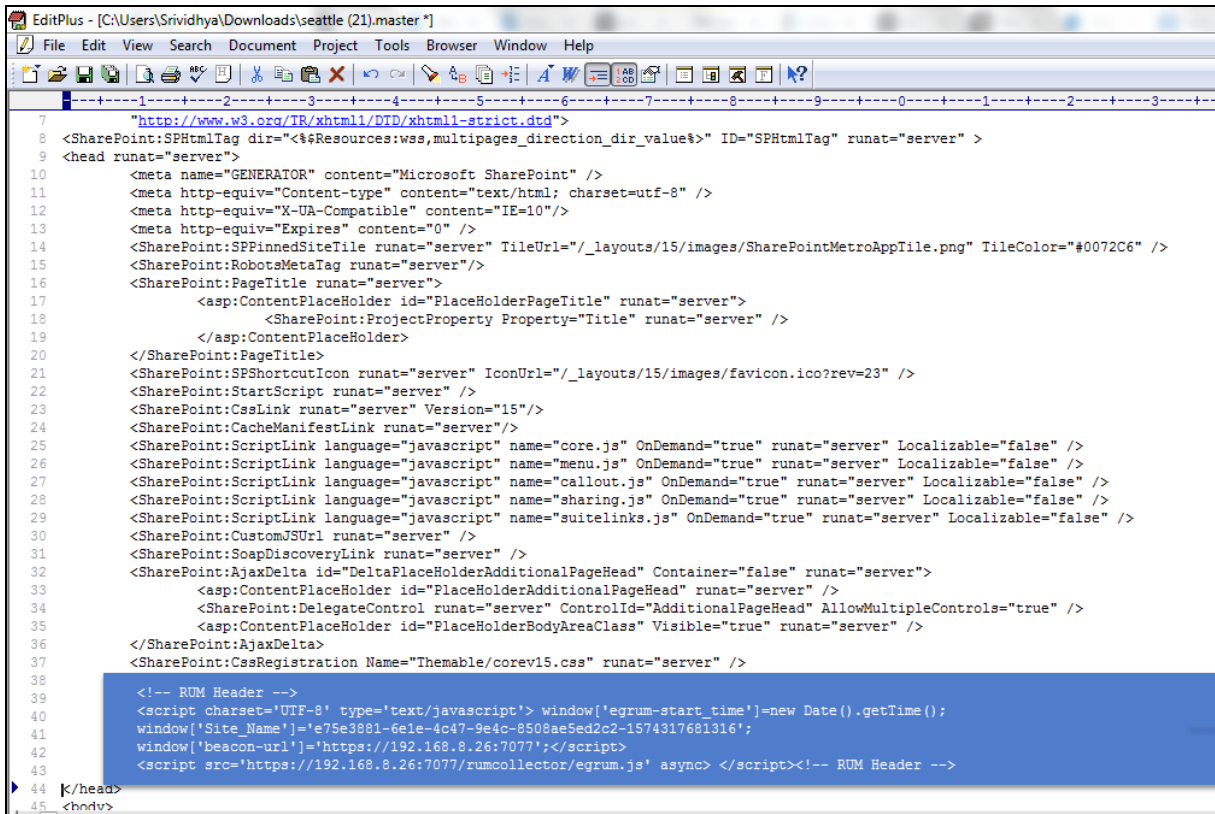


Figure 3.82: Downloading a copy of the master file

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 3.83).



```

7      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
8      <SharePoint:SPHtmlTag dir="<%=Resources:wss,multipages_direction_dir_value%>" ID="SPHtmlTag" runat="server" >
9      <head runat="server">
10         <meta name="GENERATOR" content="Microsoft SharePoint" />
11         <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
12         <meta http-equiv="X-UA-Compatible" content="IE=10"/>
13         <meta http-equiv="Expires" content="0" />
14         <SharePoint:SPFinnedSiteFile runat="server" TitleUrl="/_layouts/15/images/SharePointMetroAppTile.png" TileColor="#0072C6" />
15         <SharePoint:RobotsMetaTag runat="server"/>
16         <SharePoint:PageTitle runat="server">
17             <asp:ContentPlaceHolder id="PlaceHolderPageTitle" runat="server">
18                 <SharePoint:ProjectProperty Property="Title" runat="server" />
19             </asp:ContentPlaceHolder>
20         </SharePoint:PageTitle>
21         <SharePoint:SPShortcutIcon runat="server" IconUrl="/_layouts/15/images/favicon.ico?rev=23" />
22         <SharePoint:StartScript runat="server" />
23         <SharePoint:CssLink runat="server" Version="15"/>
24         <SharePoint:CacheManifestLink runat="server"/>
25         <SharePoint:ScriptLink language="javascript" name="core.js" OnDemand="true" runat="server" Localizable="false" />
26         <SharePoint:ScriptLink language="javascript" name="menu.js" OnDemand="true" runat="server" Localizable="false" />
27         <SharePoint:ScriptLink language="javascript" name="callout.js" OnDemand="true" runat="server" Localizable="false" />
28         <SharePoint:ScriptLink language="javascript" name="sharing.js" OnDemand="true" runat="server" Localizable="false" />
29         <SharePoint:ScriptLink language="javascript" name="suitelinks.js" OnDemand="true" runat="server" Localizable="false" />
30         <SharePoint:CustomJSUrl runat="server" />
31         <SharePoint:SoapDiscoveryLink runat="server" />
32         <SharePoint:AjaxDelta id="DeltaPlaceHolderAdditionalPageHead" Container="false" runat="server">
33             <asp:ContentPlaceHolder id="PlaceHolderAdditionalPageHead" runat="server" />
34             <SharePoint:DelegateControl runat="server" ControlId="AdditionalPageHead" AllowMultipleControls="true" />
35             <asp:ContentPlaceHolder id="PlaceHolderBodyAreaClass" Visible="true" runat="server" />
36         </SharePoint:AjaxDelta>
37         <SharePoint:CssRegistration Name="Themable/corev15.css" runat="server" />
38
39         <!-- RUM Header -->
40         <script charset="UTF-8" type="text/javascript"> window['egrum-start_time']=new Date().getTime();
41         window['Site_Name']='e75e3881-6e1e-4c47-9e4c-8508ae5ed2c2-1574317681316';
42         window['beacon-url']='https://192.168.8.26:7077';</script>
43         <script src='https://192.168.8.26:7077/rumcollector/egrum.js' async> </script><!-- RUM Header -->
44     </head>
45     <body>

```

Figure 3.83: 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 3.82. 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.

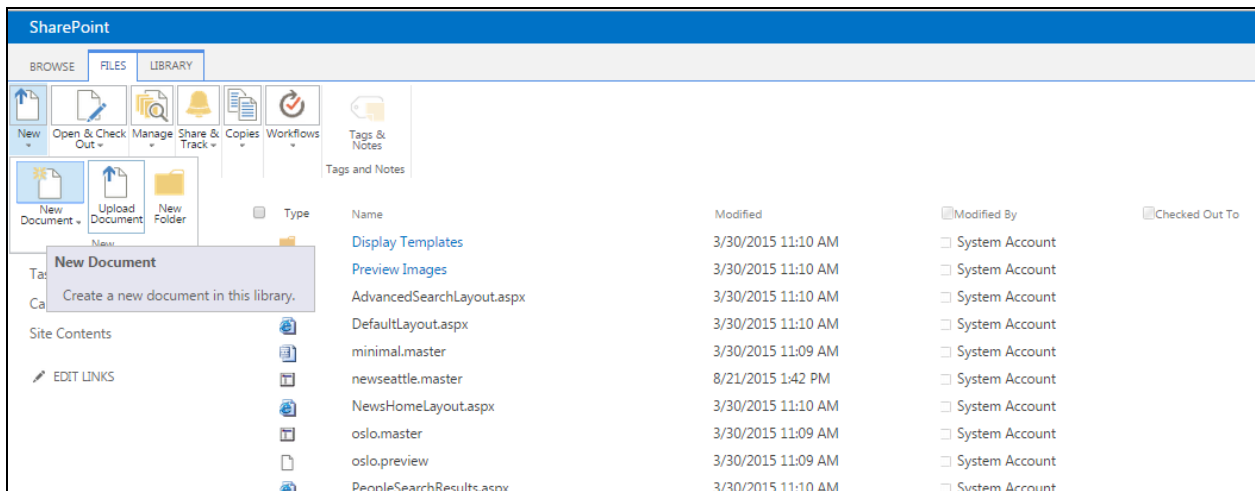


Figure 3.84: Uploading the modified master file

- Figure 3.85 will then appear. Use the **Choose File** button in Figure 3.85 to select the master file to be uploaded. Provide appropriate comments in the **Version Comments** text box, and click the **OK** button.

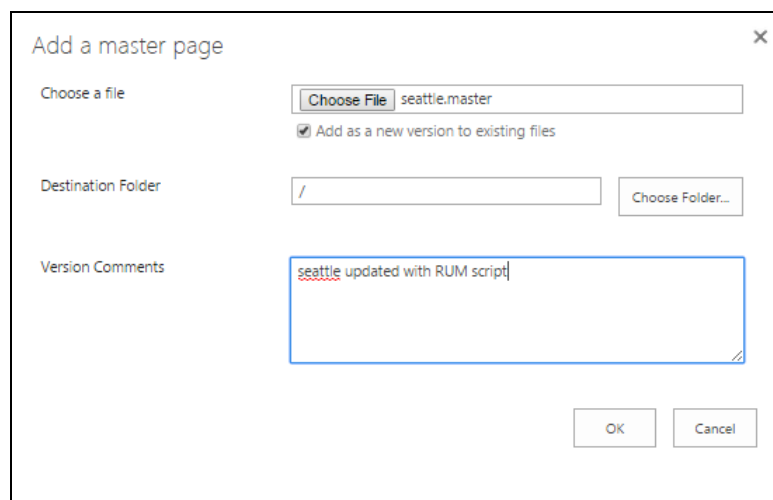


Figure 3.85: Choosing the modified master file for uploading

- When Figure 3.86 appears, provide a **Description** and click the **Save** button therein.

Master Page Gallery - seattle.master

EDIT

Commit Clipboard Actions

The document was uploaded successfully and is checked out to you. Check that the fields below are correct and that all required fields are filled out. The file will not be accessible to other users until you check in.

Content Type: Master Page
Create a new master page.

Name: seattle.master

Title: seattle

Description: first try in this app

UI Version: ☐ 4 ☒ 15

Default CSS File: corev15app.css

Version: 4.0
Created at 3/30/2015 11:09 AM by System Account
Last modified at 8/21/2015 1:56 PM by System Account

Save Cancel

Figure 3.86: Saving the changes to the master file

- 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 3.87).

Display Templates	3/30/2015 11:10 AM	☐ System Account	
Preview Images	3/30/2015 11:10 AM	☐ System Account	
AdvancedSearchLayout.aspx	3/30/2015 11:10 AM	☐ System Account	15
DefaultLayout.aspx	3/30/2015 11:10 AM	☐ System Account	15
minimal.master	3/30/2015 11:09 AM	☐ System Account	4
newseattle.master	8/21/2015 1:42 PM	☐ System Account	15
NewsHomeLayout.aspx	3/30/2015 11:10 AM	☐ System Account	15
oslo.master	3/30/2015 11:09 AM	☐ System Account	15
oslo.preview	3/30/2015 11:09 AM	☐ System Account	15
PeopleSearchResults.aspx	3/30/2015 11:10 AM	☐ System Account	15
PublishingImageRenditions.xml	3/30/2015 11:11 AM	☐ System Account	15
ReportCenterLayout.aspx	3/30/2015 11:10 AM	☐ System Account	15
SearchMain.aspx	3/30/2015 11:10 AM	☐ System Account	15
SearchResults.aspx	3/30/2015 11:10 AM	☐ System Account	15
☒ seattle.master	8/21/2015 1:56 PM	☒ System Account	15
seattle.preview	3/30/2015 11:09 AM	☐ System Account	15
TabViewPageLayout.aspx	3/30/2015 11:10 AM	☐ System Account	15
v4.master	3/30/2015 11:09 AM	☐ System Account	4
welcomelayout2.aspx	3/30/2015 11:10 AM	☐ System Account	15

Figure 3.87: 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 3.88).

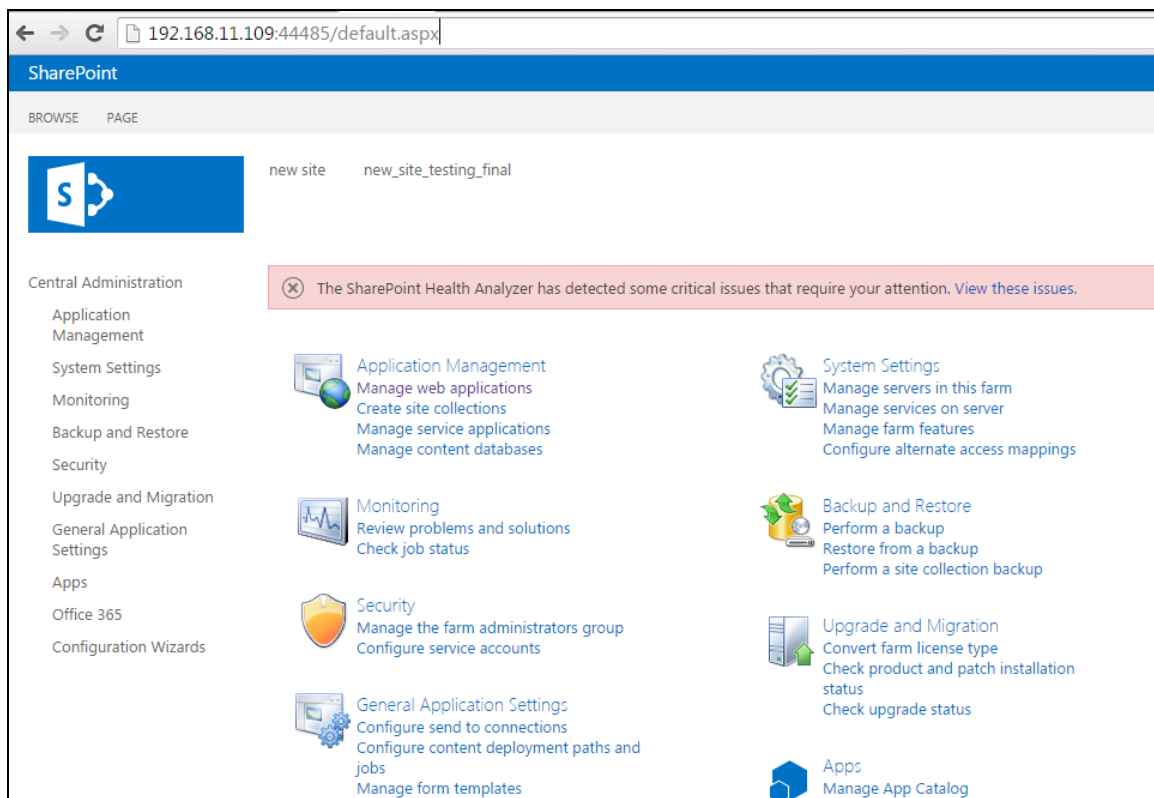


Figure 3.88: Choosing to manage web applications

18. When 3.4.7 appears, select the Sharepoint web application into which you have injected the Javascript code snippet.

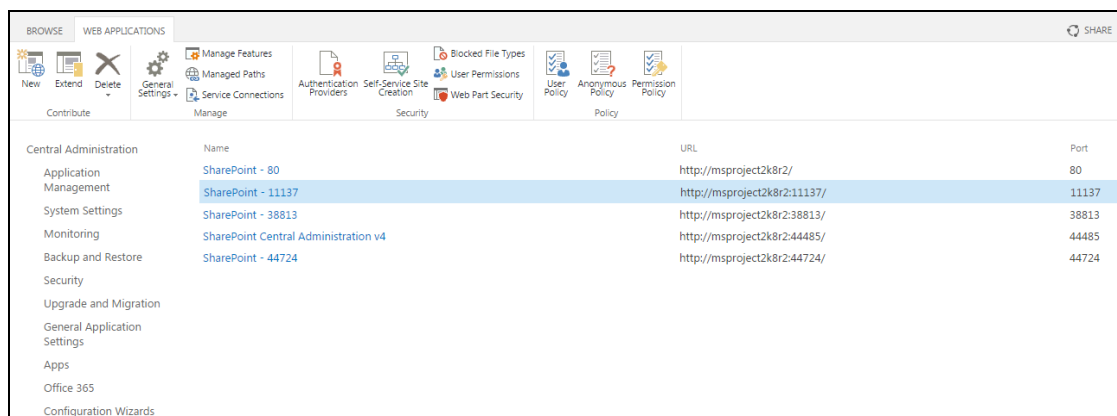


Figure 3.89: Selecting the Sharepoint web application to be monitored

19. From the tool bar at the top of Figure 3.89, click on the **General Settings** tool and then pick the **General Settings** option from the menu that appears (see Figure 3.90).

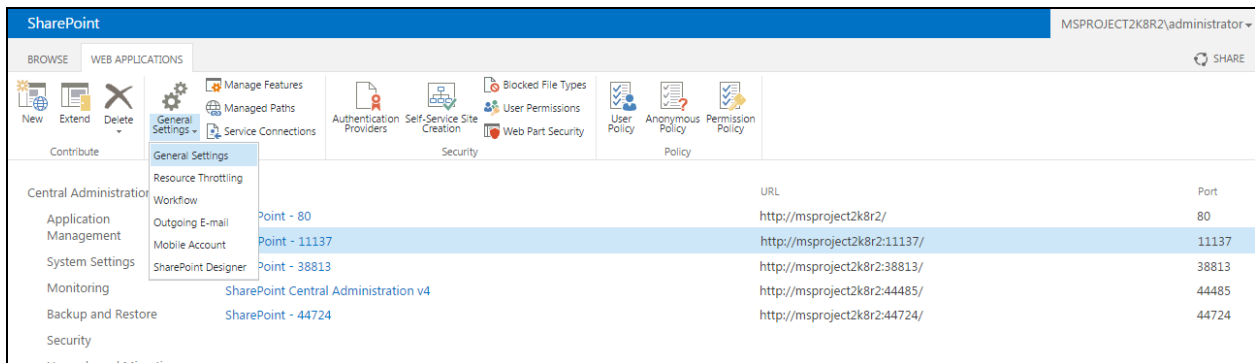


Figure 3.90: Selecting the 'General Settings' option

20. Once Figure 3.91 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**.

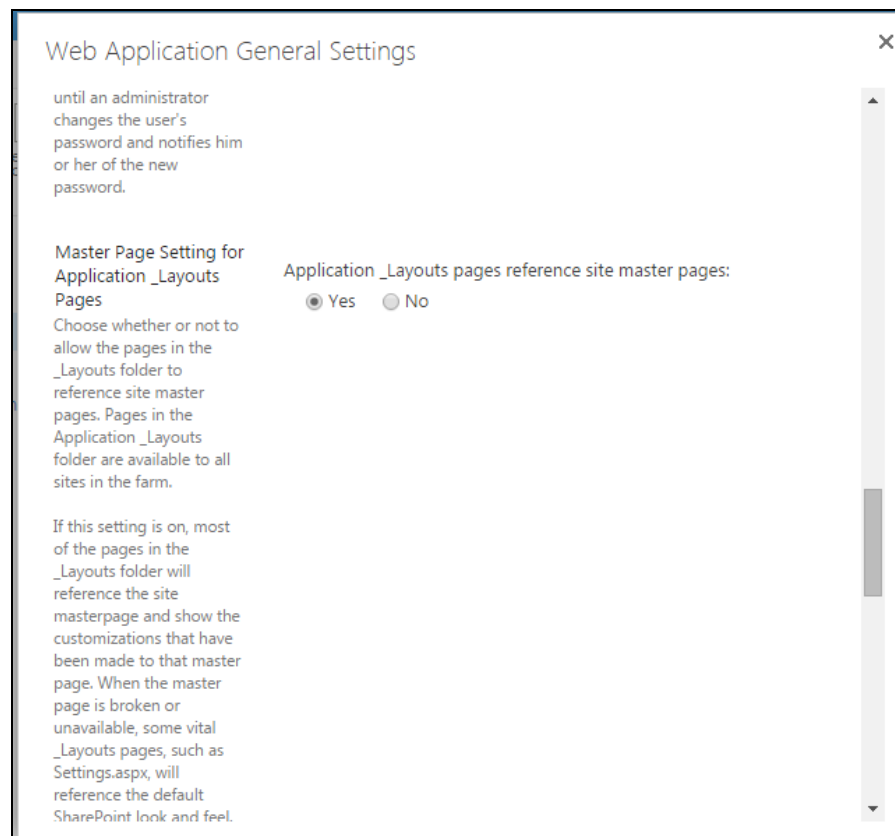


Figure 3.91: Setting the 'Application –Layouts . . .' flag to Yes

Note:

The illustrated example above takes the case of a web application on Sharepoint 2013 that is to be monitored by eG RUM. 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.

3.4.8 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 3.92).

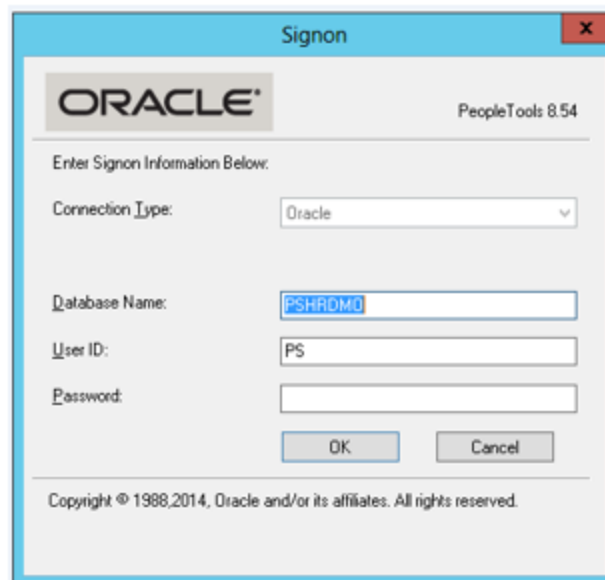


Figure 3.92: Logging into PeopleTools

2. Open the Application Designer. When Figure 3.93 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.

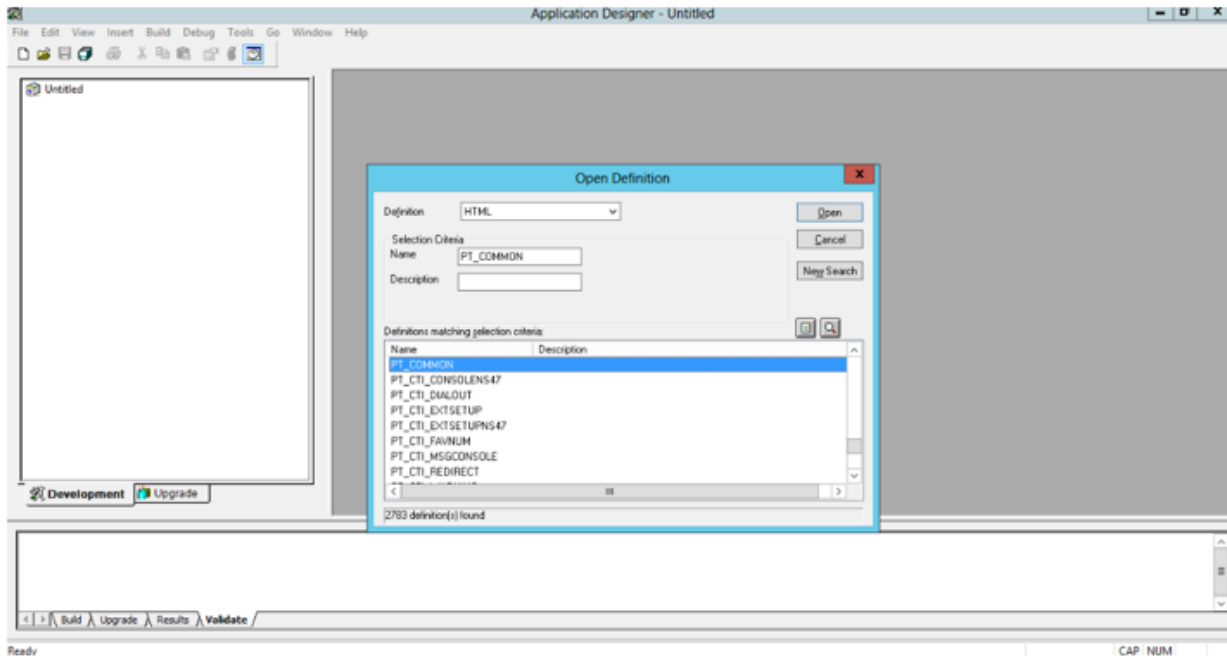


Figure 3.93: Searching for an HTML definition

3. The definitions that match the specified criteria will then be listed in the **Definitions matching selection criteria** list box (see Figure 3.93). Select the **PT_COMMON** (or **PT_COPYURL**, as the case may be) definition from the list and edit it.
4. Figure 3.94 will then appear. Here, paste the JavaScript code snippet as depicted.

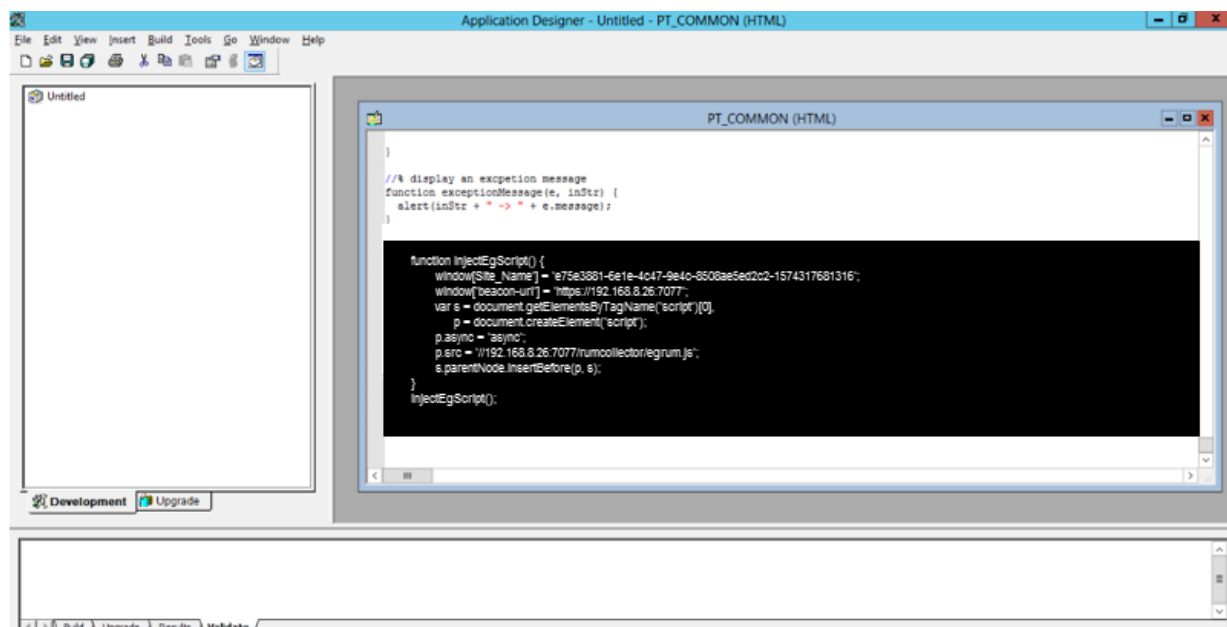


Figure 3.94: Inserting the JavaScript code snippet in the HTML definition

5. Finally, save the changes.

3.4.9 Enabling RUM for Atlassian Confluence

Atlassian Confluence is content collaboration software that changes how modern teams work. Teams can use Confluence to create, share, and collaborate on projects from one place. Confluence can also enable teams to publish, organize, and access company information in one central location. Using confluence, teams can also capture, store, and grow collective knowledge from a single place.

To measure user experience with Confluence, the RUM code snippet has to be inserted into each of the Confluence pages that teams access. This can be achieved in one of the following ways:

- By editing the Atlassian Confluence velocity template;
- By applying the Atlassian plugin and then editing the velocity template

3.4.9.1 Editing the Velocity Template to RUM-Enable Confluence

Velocity is a server-side template language used by Confluence to render page content. Velocity allows Java objects to be called alongside standard HTML.

To RUM-enable the Confluence pages, you should modify the velocity template, so that it includes the RUM code snippet. The template will then automatically inject the code snippet is every page it renders.

The detailed procedure for this is as follows:

1. The first step is to copy the RUM script to be injected. For this, login to the eG administrative interface and add a *Real User Monitor* component using the Add/Modify page (Infrastructure - > Components -> Add/Modify).
2. Once the component is added, the RUM script that should be included in the responses will be displayed. Copy this script to a text file. **Note that this code snippet is case-sensitive. It is therefore recommended that you copy the code as is to the text file, instead of manually retyping it.**

To instrument your html pages use the eG javascript agent

Download this if you have a Java web application that emits HTML to the front-end. Examples: JSF, JSP, Groovy:	egrum.jar	Download
Download this if you have a .NET web application such as ASP.NET MVC or WebForms:	Eg.RUM.dll	Download
Download this if you can edit the source code of your web pages. Examples: Plain HTML, JSP with embedded HTML, PHP with embedded HTML:	egrum.js	Download
Download this if you want to embed this egrum.js into packaged applications such as SAP Fiori:	Code snippet + egrum.js	Download

Select the source file type on which eG RUM JavaScript code snippet to be included

☒ HTML
 ☐ Javascript

Include this line in your pages (html/jsp/aspx) immediately after the head tag

```
<!-- RUM Header -->
<script charset='UTF-8' type='text/javascript'> window['egrum-start_time'] = new Date().getTime(); window['Site_Name'] = 'e75e3881-6e1e-4c47-9e47-8508ae5ed2c2-1574317681316'; window['beacon-url'] = 'https://192.168.8.26:7077';
</script>
<script src='https://192.168.8.26:7077/rumcollector/egrum.js' async> </script>
```

Figure 3.95: The RUM script that is to be injected into the HTTP responses

3. Edit the **<CONFLUENCE_INSTALL_DIR>\Atlassian\Confluence\confluence\decorators\includes\header.vm** file.
4. Insert the RUM code snippet in the text file to the header.vm file, after the meta tags, as indicated below:

```
<!-- RUM Header --> <script charset='UTF-8' type='text/javascript'> window['egrum-start_time'] = new Date().getTime(); window['Site_Name'] = 'e75e3881-6e1e-4c47-9e47-8508ae5ed2c2-15714317681316'; window['beacon-url'] = 'https://192.168.8.26:7077'; </script> <script src='http://192.168.8.26:7077/rumcollector/egrum.js' async> </script> <!-- RUM Header -->
```

5. Save the file.
6. Next, proceed to flush all page caches. For this, first, login to Atlassian Confluence.
7. When Figure 2 appears, click on the **Confluence** logo at the left, top corner of the window to view the toolbar. Then, click on the **Gear** icon on the toolbar and pick the **General configuration** option.

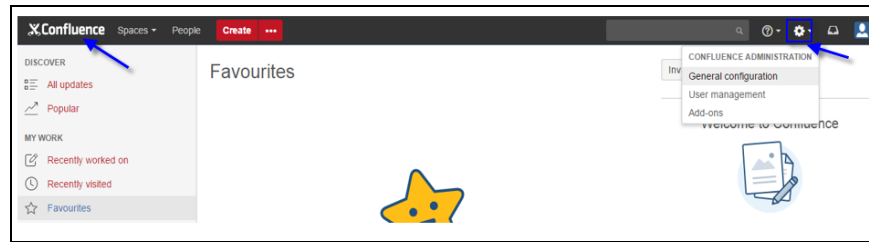


Figure 3.96: Selecting the General Configuration option

8. When Figure 3.97 appears, click on the **Cache Management** option in the left panel.

ADMINISTRATION	Content Links (Outgoing)	0%	Unknown	0 / 1000	0	0 / 0 / 0	Adjust Size
System Information	Content Links (Referral)	0%	Unknown	0 / 1000	0	0 / 0 / 0	Adjust Size
Backup & Restore	Content Links (Trackback)	0%	Unknown	0 / 1000	0	0 / 0 / 0	Adjust Size
Content Indexing	Content Objects	0%	83%	23 / 5000	Unknown	214 / 41 / 0	Adjust Size
Mail Queue	Content Permission Sets	0%	Unknown	0 / 1000	0	0 / 0 / 0	Adjust Size
Scheduled Jobs	Content Permissions	0%	Unknown	0 / 1000	0	0 / 0 / 0	Adjust Size
Cache Management	Content Properties	0%	Unknown	0 / 5000	0	0 / 0 / 0	Adjust Size
License Details	Content properties	0%	75%	3 / 5000	Unknown	9 / 3 / 0	Adjust Size
Logging and Profiling							
Application Links							
Application Navigator							
Analytics							
Support Tools							
Clustering							

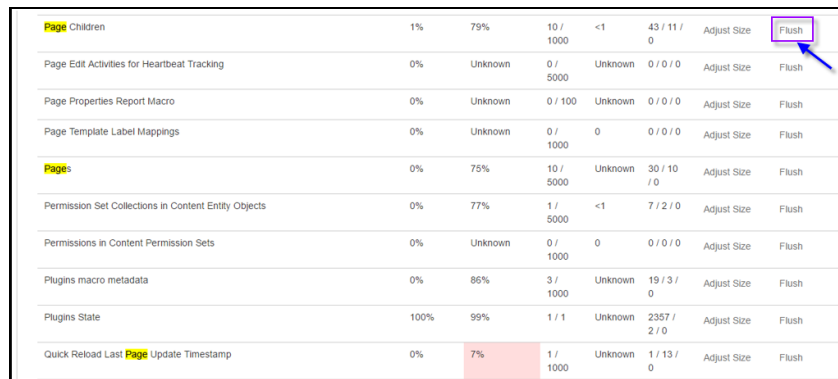
Figure 3.97: Clicking on the Cache Management option

9. Then, click on the **Show advanced view** option in the **Cache Statistics** page that appears in the right panel (see Figure 3.98).

Confluence administration			
CONFIGURATION	Cache Statistics		
General Configuration	This page displays statistics about Confluence's internal caches. You may wish to tune cache sizes and expiration policies to improve memory usage or performance. More about Cache Statistics		
Further Configuration	Show advanced view		
Backup Administration			
Manage Referrers			
Languages			
Shortcut Links			
External Gadgets			
Global Templates and Blueprints			
	Cache Name	Capacity Utilisation	Effectiveness
	Analytics Settings	0%	99%
	Application Links	100%	92%
	Attachment Download Paths	0%	87%
			Flush Entries
			Flush
			Not flushable
			Flush

Figure 3.98: Clicking on the Show advanced view link

10. To flush the page caches, click the **Flush** links that correspond to all page-related caches (see Figure 3.99).



Page Children	1%	79%	10 / 1000	<1	43 / 11 / 0	Adjust Size	Flush
Page Edit Activities for Heartbeat Tracking	0%	Unknown	0 / 5000	Unknown	0 / 0 / 0	Adjust Size	Flush
Page Properties Report Macro	0%	Unknown	0 / 100	Unknown	0 / 0 / 0	Adjust Size	Flush
Page Template Label Mappings	0%	Unknown	0 / 1000	0	0 / 0 / 0	Adjust Size	Flush
Page-	0%	75%	10 / 5000	Unknown	30 / 10 / 0	Adjust Size	Flush
Permission Set Collections in Content Entity Objects	0%	77%	1 / 5000	<1	7 / 2 / 0	Adjust Size	Flush
Permissions in Content Permission Sets	0%	Unknown	0 / 1000	0	0 / 0 / 0	Adjust Size	Flush
Plugins macro metadata	0%	86%	3 / 1000	Unknown	19 / 3 / 0	Adjust Size	Flush
Plugins State	100%	99%	1 / 1	Unknown	2357 / 2 / 0	Adjust Size	Flush
Quick Reload Last Page Update Timestamp	0%	7%	1 / 1000	Unknown	1 / 13 / 0	Adjust Size	Flush

Figure 3.99: Flushing all page-related caches

3.4.9.2 Applying an Atlassian Plugin to RUM-Enable Confluence

If, for some reason, the RUM code snippet cannot be directly inserted into the Velocity template file, then, use the broad steps outlined below to RUM-enable Atlassian Confluence:

1. Download the **egrum.jar** and include it in the classpath of Atlassian Confluence.
2. Upload the specialized plugin for Atlassian that eG Enterprise provides. This plugin includes a reference key/pointer to the **egrum.jar**.
3. Edit the Velocity template file, so that this reference key/pointer can be added to the Confluence velocity context. Once this is done, the Velocity template file will reference the **egrum.jar** using the key in the Atlassian plugin, dynamically generate the RUM script, and inject the script into the HTML pages.

The detailed procedure is provided below:

1. First, download the **egrum.jar** file to system hosting Atlassian Confluence. For this, login to the Confluence host. Then, connect to the eG manager using a browser, and login to the eG administrative interface. Next, add the Confluence site to be monitored as a *Real User Monitor* component (Infrastructure -> Components -> Add/Modify).
2. Once the component is added, click the **egrum.jar** link in Figure 3.100 that appears, to download the jar file to any location on the local host.

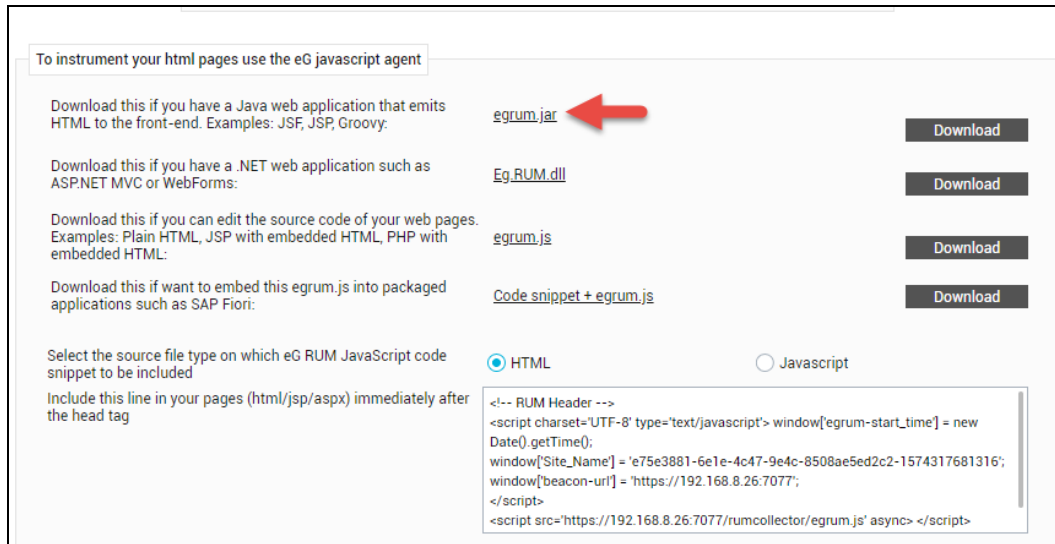


Figure 3.100: Downloading the egrum.jar file

3. Then, copy the **egrum.jar** to the <CONFLUENCE_INSTALL_DIR>\Atlassian\Confluence\lib directory.
4. Next, open the **catalina.bat** file in the <CONFLUENCE_INSTALL_DIR>\Atlassian\Confluence\bin directory, using an editor.
5. Look for the following code block in the file:

```
%_ EXECJAVA% - classpath "%CATALINA_HOME%\lib\catalina.jar" org.apache.catalina.util.ServerInfo
```

6. To add the egrum.jar to the classpath, you need to replace the above code block with the following:

```
%_ EXECJAVA% - classpath "%CATALINA_HOME%\lib\catalina.jar" "%CATALINA_HOME%\lib\egrum.jar" org.apache.catalina.util.ServerInfo
```

7. Save the file, and then restart Atlassian Confluence.
8. Next, proceed to upload the Atlassian plugin that has been bundled into the eG manager to Confluence, as an add-on. For this, first, login to Atlassian Confluence. When Figure 3.101 appears, click on the **Confluence** logo at the left, top corner of the window to view the toolbar. Then, click on the **Gear** icon on the toolbar and pick the **Add-ons** option.

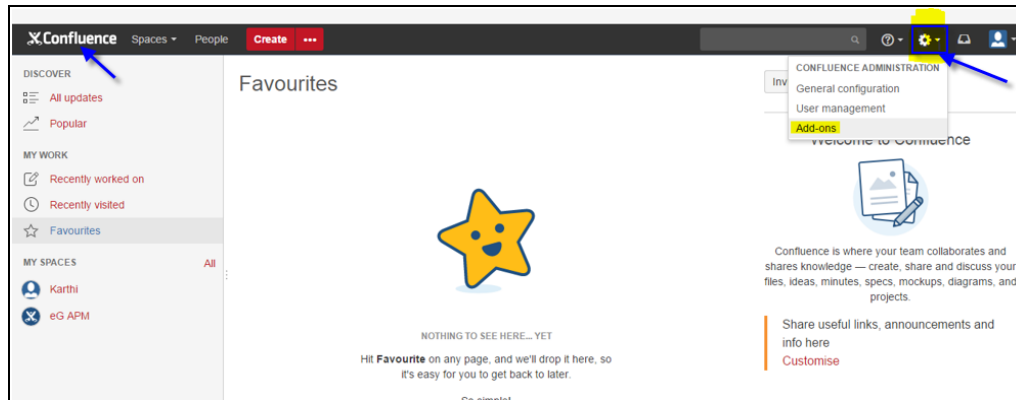


Figure 3.101: Selecting the Add-ons option

9. Figure 3.102 then appears. Click on the **Upload** add-on link in Figure 3.102.

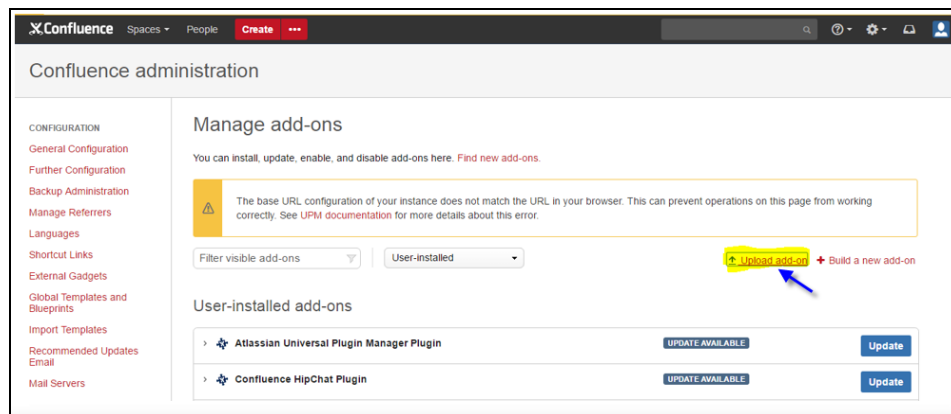


Figure 3.102: Clicking on the Upload add-on link

10. Figure 3.103 will then appear. Click the **Choose File** button therein to browse for the add-on named, *egrum-atlassian-plugin.jar*, and specify its location. This add-on will be available in the <EG_MANAGER_INSTALL_DIR>\lib\rum directory.

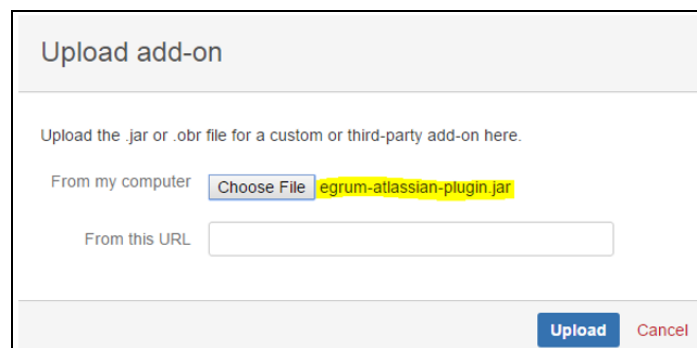


Figure 3.103: Uploading the add-on

11. Click the **Upload** button in Figure 3.103 to upload the add-on.
12. Once the upload is successful, Figure 3.104 will appear informing you of the successful upload.

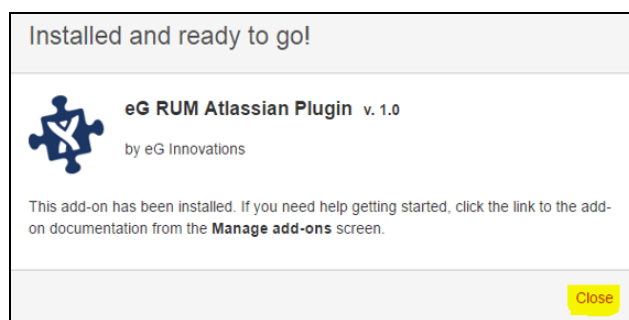


Figure 3.104: A message box confirming the successful upload

13. Click the **Close** button in Figure 3.104 and log out of Confluence.
14. Next, proceed to edit the Velocity template file. For that, open the <CONFLUENCE_INSTALL_DIR>\Atlassian\Confluence\confluence\decorators\includes\header.vm file in an editor.
15. Insert the following line in the **header.vm** file, after the meta tags, as indicated below:

```
#set($eGRumScriptAsHtml = $eGRum.getHeader())$eGRumScriptAsHtml
```
16. Save the file.
17. Next, proceed to flush all page caches. For this, first, login to Atlassian Confluence.
18. When Figure 3.105 appears, click on the **Confluence** logo at the left, top corner of the window to view the toolbar. Then, click on the **Gear** icon on the toolbar and pick the **General configuration** option.

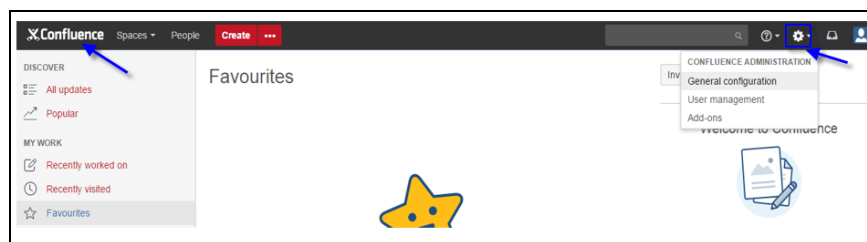


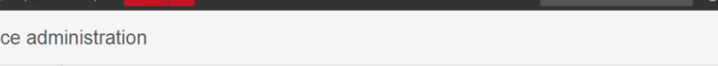
Figure 3.105: Selecting the General Configuration option

19. When Figure 3.106 appears, click on the **Cache Management** option in the left panel.

ADMINISTRATION	Content Links (Outgoing)	0%	Unknown	0 / 1000	0	0 / 0 / 0	Adjust Size
System Information	Content Links (Referral)	0%	Unknown	0 / 1000	0	0 / 0 / 0	Adjust Size
Backup & Restore	Content Links (Trackback)	0%	Unknown	0 / 1000	0	0 / 0 / 0	Adjust Size
Content Indexing	Content Objects	0%	83%	23 / 5000	Unknown	214 / 41 / 0	Adjust Size
Mail Queue	Content Permission Sets	0%	Unknown	0 / 1000	0	0 / 0 / 0	Adjust Size
Scheduled Jobs	Content Permissions	0%	Unknown	0 / 1000	0	0 / 0 / 0	Adjust Size
Cache Management	Content Properties	0%	Unknown	0 / 5000	0	0 / 0 / 0	Adjust Size
License Details	Content properties	0%	75%	3 / 5000	Unknown	9 / 3 / 0	Adjust Size
Logging and Profiling							
Application Links							
Application Navigator							
Analytics							
Support Tools							
Clustering							

Figure 3.106: Clicking on the Cache Management option

20. Then, click on the **Show advanced view** option in the **Cache Statistics** page that appears in the right panel (see Figure 3.107).



The screenshot shows the Confluence administration interface. On the left is a sidebar with a 'CONFIGURATION' section containing links like 'General Configuration', 'Further Configuration', 'Backup Administration', 'Manage Referrers', 'Languages', 'Shortcut Links', 'External Gadgets', 'Global Templates and Blueprints'. The main content area is titled 'Cache Statistics' and contains a paragraph explaining that the page displays statistics about Confluence's internal caches. Below this paragraph is a table with four columns: 'Cache Name', 'Capacity Utilisation', 'Effectiveness', and 'Flush Entries'. The table lists three cache types: 'Analytics Settings' (0% utilisation, 99% effectiveness, flushable), 'Application Links' (100% utilisation, 92% effectiveness, not flushable), and 'Attachment Download Paths' (0% utilisation, 87% effectiveness, flushable). A red rectangular box highlights the 'Show advanced view' link, and a red arrow points to it from the right.

Cache Statistics

This page displays statistics about Confluence's internal caches. You may wish to tune cache sizes and expiration policies to improve memory usage or performance. [More about Cache Statistics](#)

[Show advanced view](#)

Cache Name	Capacity Utilisation	Effectiveness	Flush Entries
Analytics Settings	0%	99%	Flush
Application Links	100%	92%	Not flushable
Attachment Download Paths	0%	87%	Flush

Figure 3.107: Clicking on the Show advanced view link

21. To flush the page caches, click the **Flush** links that correspond to all page-related caches (see Figure 3.108).

Page Children	1%	79%	10 / 1000	<1	43 / 11 / 0	Adjust Size	Flush
Page Edit Activities for Heartbeat Tracking	0%	Unknown	0 / 5000	Unknown	0 / 0 / 0	Adjust Size	Flush
Page Properties Report Macro	0%	Unknown	0 / 100	Unknown	0 / 0 / 0	Adjust Size	Flush
Page Template Label Mappings	0%	Unknown	0 / 1000	0	0 / 0 / 0	Adjust Size	Flush
Pages	0%	75%	10 / 5000	Unknown	30 / 10 / 0	Adjust Size	Flush
Permission Set Collections in Content Entity Objects	0%	77%	1 / 5000	<1	7 / 2 / 0	Adjust Size	Flush
Permissions in Content Permission Sets	0%	Unknown	0 / 1000	0	0 / 0 / 0	Adjust Size	Flush
Plugins macro metadata	0%	86%	3 / 1000	Unknown	19 / 3 / 0	Adjust Size	Flush
Plugins State	100%	99%	1 / 1	Unknown	2357 / 2 / 0	Adjust Size	Flush
Quick Reload Last Page Update Timestamp	0%	7%	1 / 1000	Unknown	1 / 13 / 0	Adjust Size	Flush

Figure 3.108: Flushing all page-related caches

3.4.10 Instrumenting a Web Site or Web Application Using a NetScaler Rewrite Policy

Rewrite refers to the rewriting of some information in the requests or responses handled by the NetScaler appliance. When you enable the rewrite feature, NetScaler can modify the headers and body of HTTP requests and responses.

Where NetScaler is used to load-balance requests to a web application, you can configure a rewrite policy on NetScaler to automatically inject the RUM script into HTTP responses from the target application.

The steps in this regard are as follows:

1. The first step is to copy the RUM script to be injected. For this, login to the eG administrative interface and add a *Real User Monitor* component using the Add/Modify page (Infrastructure - > Components -> Add/Modify).
2. Once the component is added, the RUM script that should be included in the responses will be displayed. Copy this script to a text file. **Note that this code snippet is case-sensitive. It is therefore recommended that you copy the code as is to the text file, instead of manually retyping it.**

To instrument your html pages use the eG javascript agent

Download this if you have a Java web application that emits HTML to the front-end. Examples: JSF, JSP, Groovy:	egrum.jar	Download
Download this if you have a .NET web application such as ASP.NET MVC or WebForms:	Eg.RUM.dll	Download
Download this if you can edit the source code of your web pages. Examples: Plain HTML, JSP with embedded HTML, PHP with embedded HTML:	egrum.js	Download
Download this if want to embed this egrum.js into packaged applications such as SAP Fiori:	Code snippet + egrum.js	Download

Select the source file type on which eG RUM JavaScript code snippet to be included

☒ HTML
 ☐ Javascript

Include this line in your pages (html/jsp/aspx) immediately after the head tag

```
<!-- RUM Header -->
<script charset='UTF-8' type='text/javascript'> window['egrum-start_time'] = new Date().getTime();
window['Site_Name'] = 'e75e3881-6e1e-4c47-9e4c-8508ae5ed2c2-1574317681316';
window['beacon-uri'] = 'https://192.168.8.26:7077';
</script>
<script src='https://192.168.8.26:7077/rumcollector/egrum.js' async> </script>
```

Figure 3.109: The RUM script that is to be injected into the HTTP responses

3. A sample RUM script is provided below:

```
<script charset='UTF-8' type='text/javascript'> window['egrum-start_time'] = new Date().getTime();

window['Site_Name'] = '88972966-59d2-4fb3-b01c-bed5b29b4094-1487670703991';
```

```
window['beacon-url'] = 'http://xyz.com';  
  
</script>  
  
<script src='http://xyz.com/rumcollector/egrum.js' async> </script>
```

4. Edit the RUM script in the manner described below:

- Split the code snippet into two script blocks, as shown below:

```
<script charset='UTF-8' type='text/javascript'> window['egrum-start_time'] = new  
Date().getTime();window['Site_Name'] = '88972966-59d2-4fb3-b01c-bed5b29b4094-  
1487670703991'; window['beacon-url'] = 'http://xyz.com';</script>
```

```
<script src='http://xyz.com/rumcollector/egrum.js' async> </script>
```

- Then, concatenate both the script blocks using '+', as shown below:

```
<script charset='UTF-8' type='text/javascript'> window['egrum-start_time'] = new  
Date().getTime();window['Site_Name'] = '88972966-59d2-4fb3-b01c-bed5b29b4094-  
1487670703991'; window['beacon-url'] = 'http://xyz.com';</script>+<script  
src='http://xyz.com/rumcollector/egrum.js' async> </script>
```

5. Next, enable the **Rewrite** feature on the NetScaler appliance. For this, follow the steps detailed below:

- Connect to the NetScaler Admin console via a browser.
- Login to the console. Upon login, the **Configuration** tab page will open by default. From the **System** menu in the left panel of the console, select the **Settings** option. The right panel will then change to display a **Settings** page (see Figure 3.110).

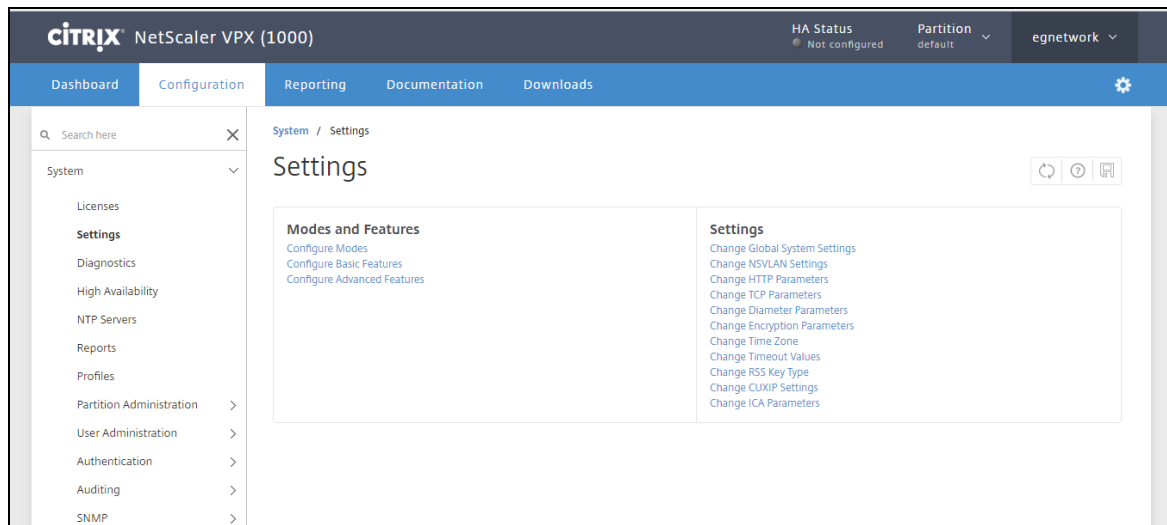


Figure 3.110: Selecting the Configure Basic Features option

- From the **Modes and Features** menu in the **Settings** page (see Figure 3.110), select the **Configure Basic Features** option. Figure 3.111 will then appear:

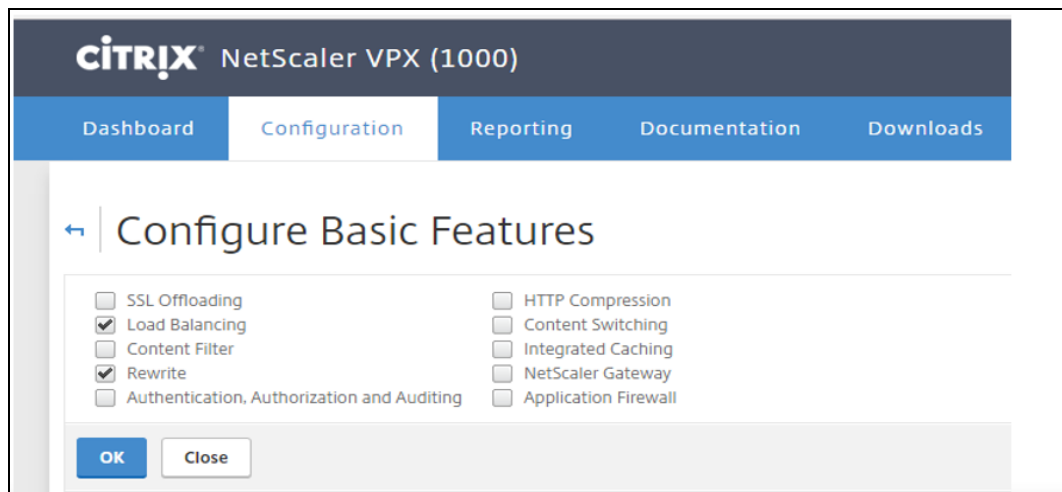


Figure 3.111: The Configure Basic Features page

- Next, enable rewrite by selecting the **Rewrite** check box, as shown by Figure 3.111. Click **OK** in Figure 3.111 to save the changes. You will now return to Figure 3.110. Close the **System** menu in the left panel of Figure 3.110.
6. Now, proceed to define a rewrite action. This is required to insert the eG RUM code snippet in the body of the HTTP responses sent by the target application. For this, follow the steps below:

- First expand the **AppExpert** menu in the left panel of Figure 3.110. From the **AppExpert** menu, select the **Rewrite** sub-menu, and then pick the **Actions** option. The right panel will then change to display the rewrite actions that pre-exist. To add a new rewrite action, click the **Add** button in the right panel of Figure 3.112.

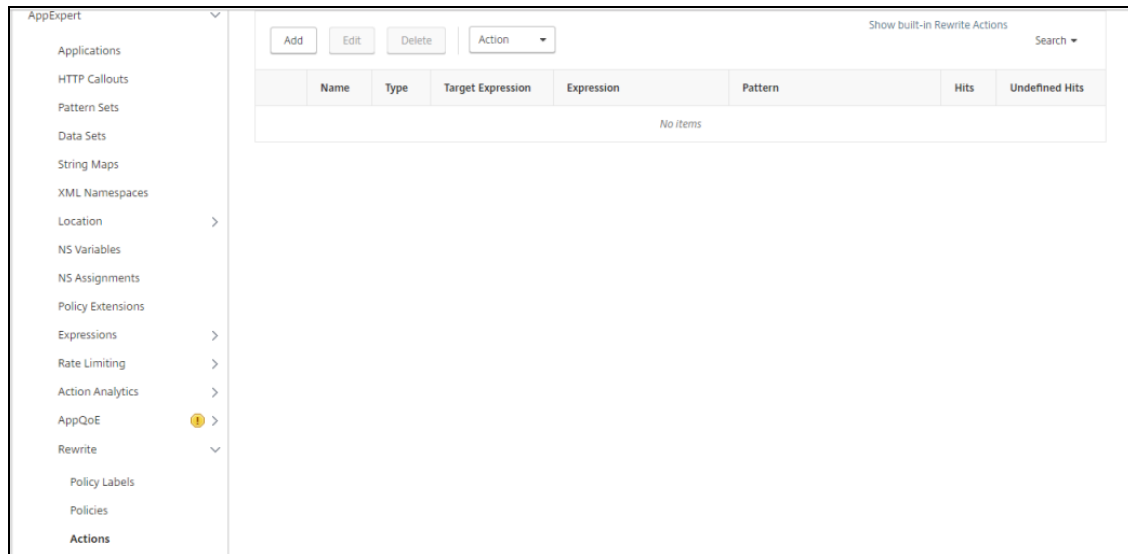


Figure 3.112: Clicking on the Add button to add a new rewrite action

- Figure 3.113 and Figure 3.114 will then appear.

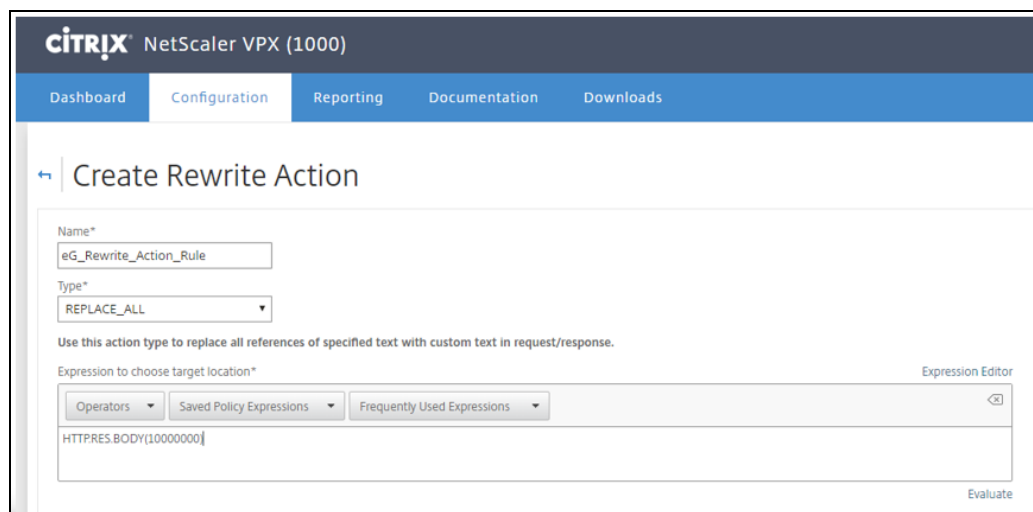
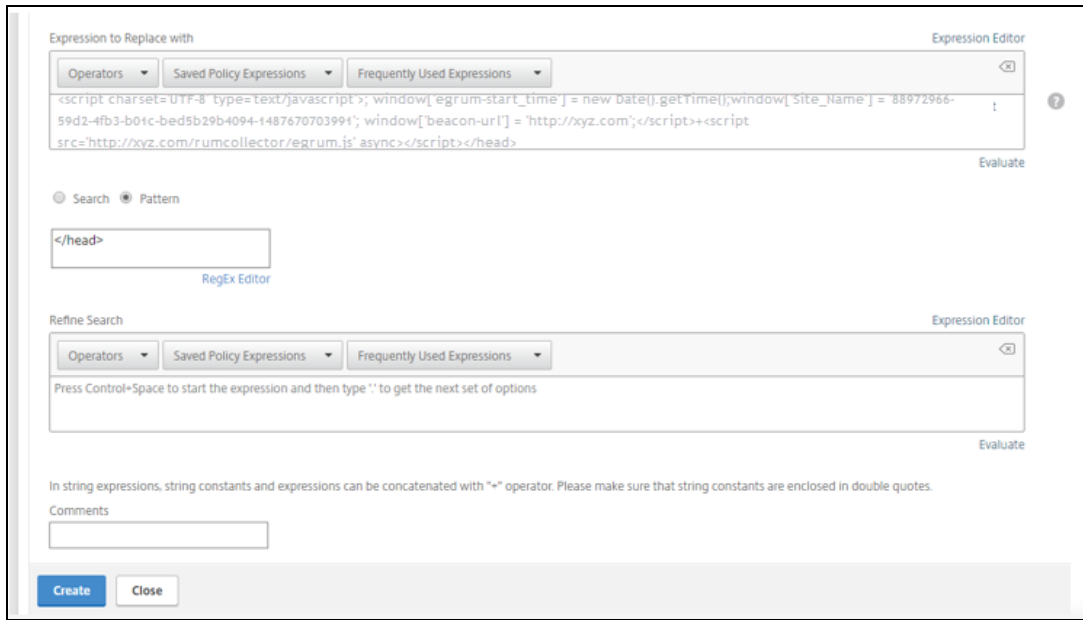


Figure 3.113: Adding a rewrite action to replace the body of the HTTP response



Expression to Replace with

Expression Editor

Operators Saved Policy Expressions Frequently Used Expressions

```
<script charset=UTF-8 type=Text/javascript>; window[ 'egrum-start_time' ] = new Date().getTime();window[ 'Site_Name' ] = 88972966-59d2-4fb3-b01c-bed5b29b4094-1487670703994; window[ 'beacon-url' ] = 'http://xyz.com';</script>+<script src='http://xyz.com/rumcollector/egrum.js' async></script></head>
```

Evaluate

Search Pattern

RegEx Editor

Refine Search

Expression Editor

Operators Saved Policy Expressions Frequently Used Expressions

Press Control+Space to start the expression and then type '' to get the next set of options

Evaluate

In string expressions, string constants and expressions can be concatenated with "+" operator. Please make sure that string constants are enclosed in double quotes.

Comments

Create Close

Figure 3.114: Replacing the body of the HTTP response with the RUM code snippet (applicable to NetScaler appliances before v12)

- Using Figure 3.113 and Figure 3.114, do the following:
 - Provide a **Name** for the policy.
 - Set **Type** as *REPLACE_ALL*. This option will replace all references of a specified text with custom text in request/response.
 - In the **Expression to choose target location** text area, enter the following:

HTTP.RES.BODY(10000000)

This indicates that changes need to be made to the body of the HTTP response.

- Then, in the **Expression to Replace with** text area, enter what needs to go into the body of the HTTP response. In our case, the RUM code snippet should go into the body of the HTTP response. The contents of this text area should be in the following format:

"<RUM code snippet></head>"

where, the RUM code snippet is the text file that you saved in at step 4 above. For example the code should be specified as shown below:

```
<script charset='UTF-8' type='text/javascript'> window['egrum-start_time'] = new
Date().getTime();window['Site_Name'] = '88972966-59d2-4fb3-b01c-bed5b29b4094-
1487670703991'; window['beacon-url'] = 'http://xyz.com';</script>+<script
src='http://xyz.com/rumcollector/egrum.js' async> </script></head>
```

- If you are using a NetScaler appliance that is of a version prior to v12, then select the **Pattern** option and type `</head>` in the text area below (see Figure 3.114). On the other hand, if you are using a NetScaler appliance that is of a v12 (or above), then select the **Search** option, pick **Text** from the drop-down list below, and type `</head>` in the text area (see Figure 3.115).

Figure 3.115: Replacing the body of the HTTP response with the RUM code snippet (applicable to NetScaler appliances of v12 and above)

- Finally, click the **Create** button to create the new rewrite action.
7. Then, proceed to create a rewrite policy. A rewrite policy consists of a rule and action. The rule determines the traffic on which rewrite is applied and the action determines the action to be taken by the NetScaler. Using the rewrite policy, we will be instructing NetScaler to not insert the RUM code snippet in the responses, if the content-type of the HTTP responses is not text/html. To achieve this, follow the steps discussed hereunder:
 - From the **AppExpert** menu in the left panel of Figure 3.111, select the **Rewrite** sub-menu, and choose the **Policies** option. The right panel will then change to display the **Rewrite Policies** page (see Figure 3.116). Click the **Add** button in the right panel to add a new rewrite policy.

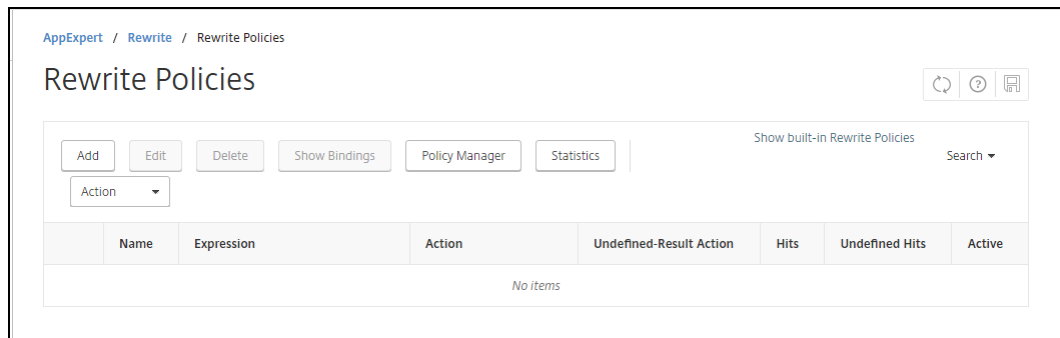


Figure 3.116: Clicking the Add button to add a new rewrite policy

- Figure 3.117 will then appear.

Figure 3.117: Creating a new rewrite policy

- In Figure 3.117, specify the following:
 - Provide a **Name** to the new rewrite policy.
 - Select the rewrite **Action** to which this new policy applies. Here, you will have to select the name of the rewrite action we created at step 6 above.
 - From the **Underlined-Result Action** drop-down, pick **NOREWRITE**.
 - Then, in the **Expression** text area, enter the following:

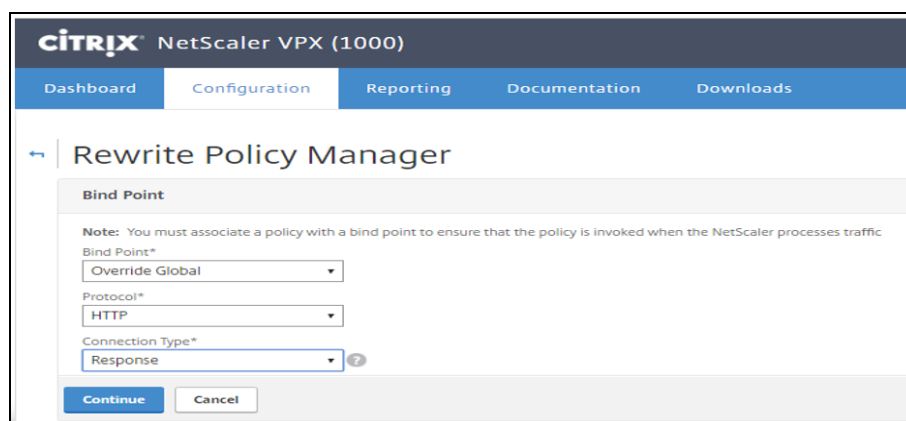
HTTP.RES.HEADER("Content-Type").CONTAINS("text/html")
 - Finally, click **Create** to create the new rewrite policy.

8. After creating a rewrite policy, you must bind it to put it into effect. You can bind your policy to

Global if you want to apply it to all traffic that passes through your NetScaler, or you can bind your policy to a specific virtual server or bind point to direct only that virtual server or bind point's incoming traffic to that policy. If an incoming request matches a rewrite policy, the action associated with that policy is carried out.

Let us bind our policy to Global, so that it applies to all traffic. For that, follow the steps below:

- From the **AppExpert** menu in the left panel of Figure 3.111, select the **Rewrite** menu, and then select the **Policies** option within. The right panel will then change to display Figure 3.116. Click on the **Policy Manager** button in Figure 3.116. Figure 3.118 will then appear.



The screenshot shows the Citrix NetScaler VPX (1000) interface. The top navigation bar includes 'Dashboard', 'Configuration', 'Reporting', 'Documentation', and 'Downloads'. The main heading is 'Rewrite Policy Manager'. Below this, there is a 'Bind Point' section with a note: 'Note: You must associate a policy with a bind point to ensure that the policy is invoked when the NetScaler processes traffic.' The 'Bind Point*' dropdown menu is set to 'Override Global'. The 'Protocol*' dropdown menu is set to 'HTTP'. The 'Connection Type*' dropdown menu is set to 'Response'. At the bottom of the form, there are 'Continue' and 'Cancel' buttons.

Figure 3.118: Binding the rewrite policy

- Specify the following in Figure 3.118:
 - Select *Override Global* as the **Bind Point**.
 - Select *HTTP* as the **Protocol**.
 - Choose **Response** as the **Connection Type**.
 - Then, click **Continue**.

- Figure 3.119 will then appear.

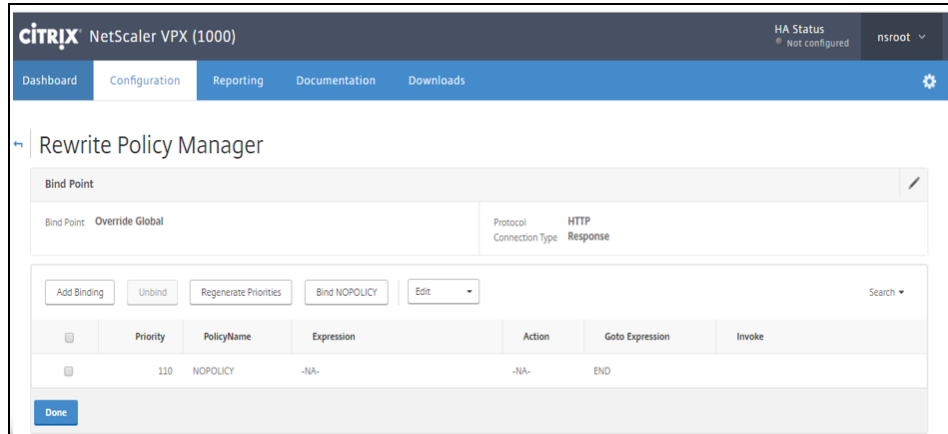


Figure 3.119: Clicking on the Add Binding button

- Click on the **Add Binding** button in Figure 3.119. Figure 3.120 will then appear.

Figure 3.120: Adding a Binding

- Specify the following in Figure 3.120:
 - From the **Select Policy** drop-down, select the rewrite policy rule you created at step 7 above.
 - Set **Priority** as *120* , **Goto Expression** as *END* , and **Invoke LabelType** as *None*.
 - Click on the **Bind** button in Figure 3.120 to add the binding.
- Figure 3.121 will then appear displaying the details of the rewrite policy rule created at step 7. Click on the **Done** button in Figure 3.121.

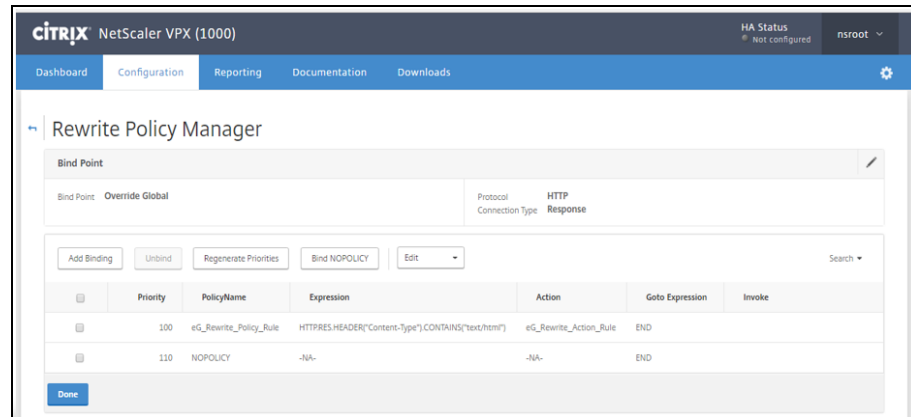


Figure 3.121: Completing the rewrite policy rule creation

3.4.11 Instrumenting a Web Application Using an F5 BIG-IP Load Balancer

Where requests to a web application are load balanced using an F5 BIG-IP load balancer, you can use the broad steps outlined below to RUM-enable that web application:

1. Copy the RUM code snippet to a text file.
2. Create an HTML Rule on the F5 BIG-IP system
3. Create an HTML Profile and associate the HTML Rule with it
4. Enable the HTML Profile on the virtual server that processes HTTP traffic to the target web application

Each of these steps have been elaborated below:

1. The first step is to copy the RUM script to be injected. For this, login to the eG administrative interface and add a *Real User Monitor* component using the Add/Modify page (Infrastructure - > Components -> Add/Modify).
2. Once the component is added, the RUM script that should be included in the responses will be displayed. Copy this script to a text file. **Note that this code snippet is case-sensitive. It is therefore recommended that you copy the code as is to the text file, instead of manually retyping it.**

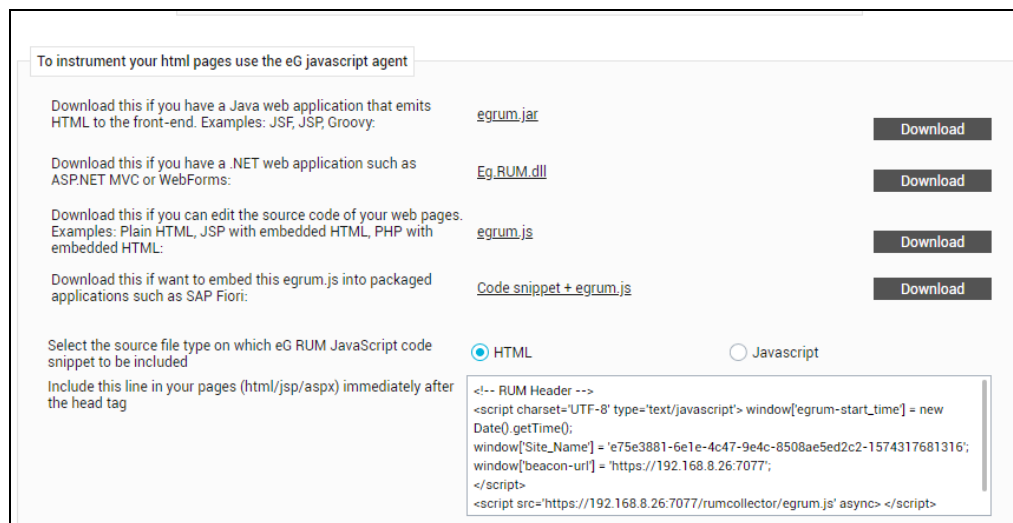


Figure 3.122: The RUM script that is to be injected into the HTTP responses

- Next, create an HTML Rule on the load balancer. An HTML rule defines the specific actions that you want the BIG-IP system to perform on a specified type of HTML content. For RUM-enabling a web application, you should create an HTML rule that searches for the `/head` tag in the HTML responses sent out by the F5 BIG-IP system and automatically inserts the RUM code snippet (that was copied to a text file at step 2 above) just before the `/head` tag in each of the responses. To create such an HTML rule, do the following:

- Login to the F5 console. Figure 3.123 will then appear.

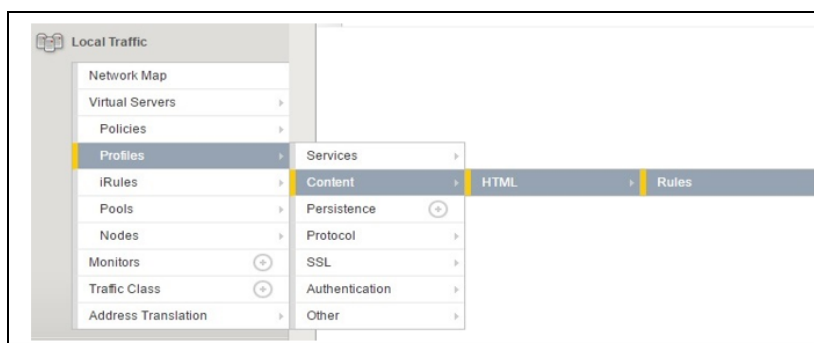


Figure 3.123: Selecting the HTML -> Rules option in the F5 console

- In the **Local Traffic** menu of Figure 3.123, follow the menu sequence, Profiles -> Content -> HTML -> Rules. Figure 3.124 will then appear.

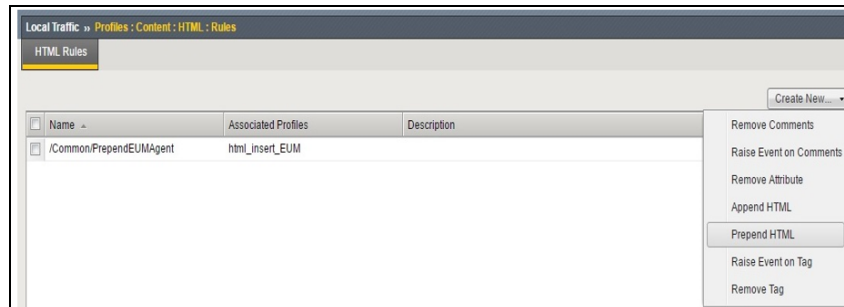


Figure 3.124: Selecting the Prepend HTML option

- Using Figure 3.124, create a **Prepend HTML Rule**. Such a rule typically matches on the specified tag name, attribute name, and attribute value, and then prepends the specified HTML content to the tag delimiter. Click on the **Create New** drop-down in the **HTML Rules** page of Figure 3.124 and select the **Prepend HTML** option. This will invoke Figure 3.125, where you will have to specify the **Rule Name**.

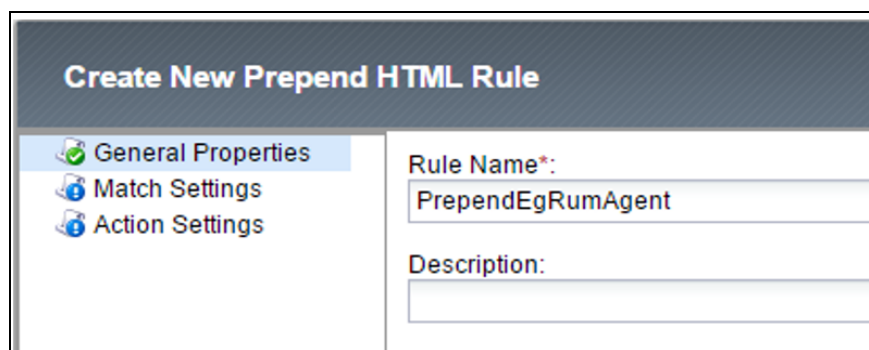
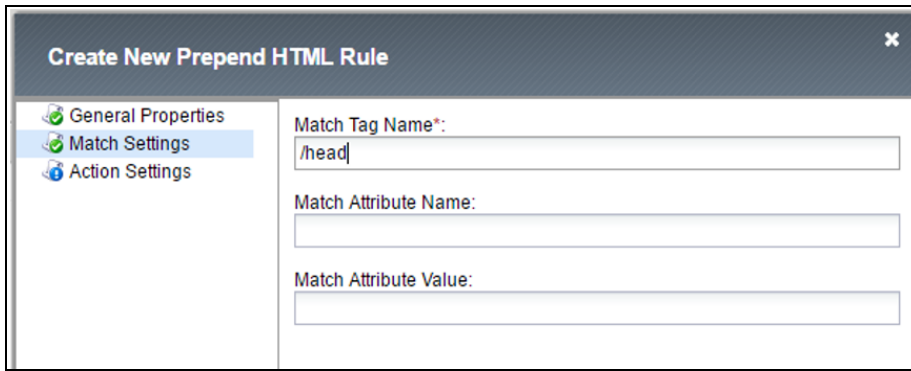


Figure 3.125: Specifying the name of the Prepend HTML rule

- Then, click the **Match Settings** option from the left panel of Figure 3.125. To instruct the rule to look for a match for the `/head` tag, specify `/head` against **Match Tag Name** in the right panel of Figure 3.126.



Create New Prepend HTML Rule

- General Properties
- Match Settings**
- Action Settings

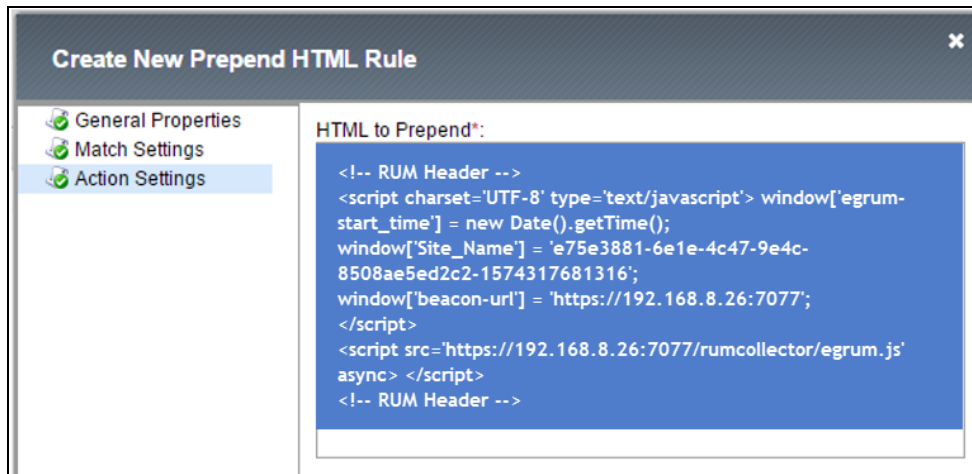
Match Tag Name*:

Match Attribute Name:

Match Attribute Value:

Figure 3.126: Specifying a Match Tag Name

- Next, choose the **Action Settings** option from the left panel of Figure 3.126. Doing so will display the **HTML to Prepend** text area in the right panel. Copy the RUM code snippet from the text file you created at step 2 above and paste it in this text area (see Figure 3.127).



Create New Prepend HTML Rule

- General Properties
- Match Settings
- Action Settings**

HTML to Prepend*:

```
<!-- RUM Header -->
<script charset='UTF-8' type='text/javascript'> window['egrum-
start_time'] = new Date().getTime();
window['Site_Name'] = 'e75e3881-6e1e-4c47-9e4c-
8508ae5ed2c2-1574317681316';
window['beacon-url'] = 'https://192.168.8.26:7077';
</script>
<script src='https://192.168.8.26:7077/rumcollector/egrum.js'
async> </script>
<!-- RUM Header -->
```

Figure 3.127: Copying the RUM code snippet to the HTML to Prepend text area

- Finally, click the **OK** button (not shown in Figure 3.127) to save the changes.
- The next step is to create an HTML profile and associate the rule you created at step 3 with the profile. When you configure an HTML profile on the BIG-IP system, the system can modify HTML content that passes through the system, according to your specifications. For this, do the following:
 - In the F5 console, invoke the **Local Traffic** menu once again. This time, follow the menu sequence, Profiles -> Content -> HTML. When Figure 3.128 appears, click on the **Create New Profile** button in Figure 3.128 to create a new HTML Profile.

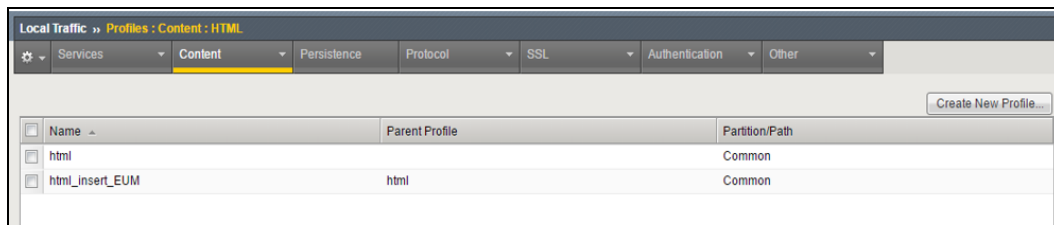


Figure 3.128: Clicking on the Create New Profile button

- Figure 3.129 will then appear. Provide a unique name for the new profile in the **Profile Name** text box of Figure 3.129. Also, from the **Parent Profile** drop-down, select */Common/html* (see Figure 3.129).



Figure 3.129: Creating a new HTML profile

- Then, click the **HTML Rules** option in the left panel of Figure 3.129. The right panel will then change as depicted by Figure 3.130. From the **Available Rules** list in the right panel (see Figure 3.130), select the HTML rule you created at step 3 above. To select, simply click on the check box alongside the rule name. Then, click the < button to transfer the selection to the **Selected Rules** list. This will associate the rule with the HTML profile being created.

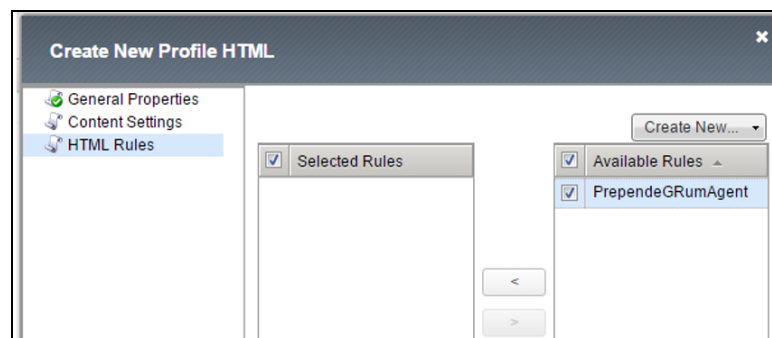


Figure 3.130: Associating the HTML Rule with the HTML Profile

- Then, click on the OK button (not shown in Figure 3.130) to save the changes to the new profile.
5. The final step is to enable the HTML profile you created at step 4 for the Virtual server that will be processing the HTML traffic to and from the target web application. For this, follow the steps below:
- In the F5 console, invoke the **Local Traffic menu**, and then follow the menu sequence, Virtual Servers -> Virtual Server List. Figure 3.131 will then appear.

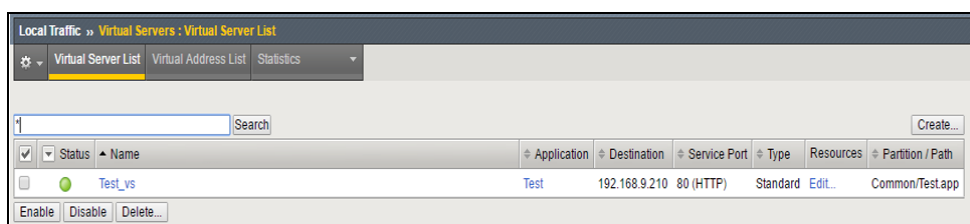


Figure 3.131: List of virtual servers

- A list of virtual servers will be displayed in Figure 3.131. Click on that virtual server on which you want the HTML profile to be enabled. This will open Figure 3.132. From the **HTML Profile** drop-down of the **Content Rewrite** section in Figure 3.132, select the profile you created at step 4 above.

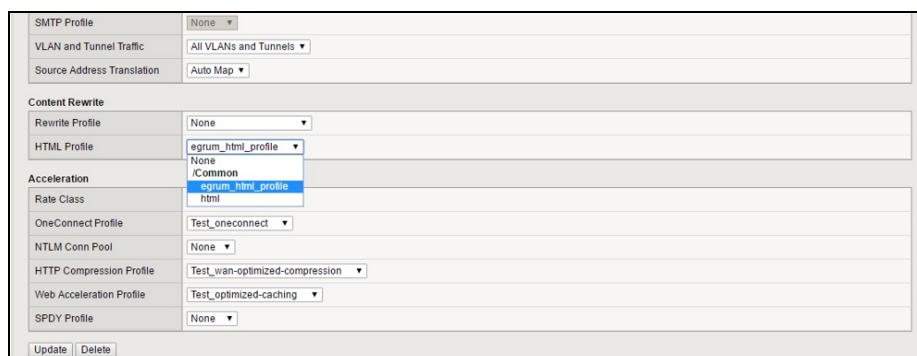


Figure 3.132: Selecting the HTML profile to be enabled on the Virtual server

- Finally, click the **Update** button in Figure 3.132 to save the changes.

Note:

Sometimes, when updating the virtual server details by clicking the **Update** button in Figure 3.132, the following error may occur:

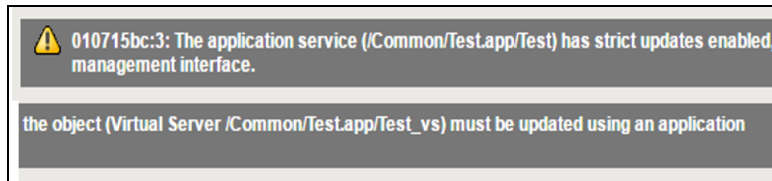


Figure 3.133: Error that appears when attempting to update the virtual server details

Such an error occurs if the application service associated with the virtual server object has **Strict Updates** enabled. If this setting is enabled, then users can control only objects that are exposed through the templates that are used by that application service.

To resolve this error, you first need to disable **Strict Updates** for the application service associated with the virtual server object. The steps to achieve this are as follows:

- Login to the F5 console.
- Follow the iApps -> Application Services menu sequence.
- Click on the **Application Service** for which you want **Strict Updates** disabled.
- Then, click on the **Properties** tab page. In this tab page, choose *Advanced* against **Application Service**, deselect the **Strict Updates** check box, and click on the **Update** button (see Figure 3.134).

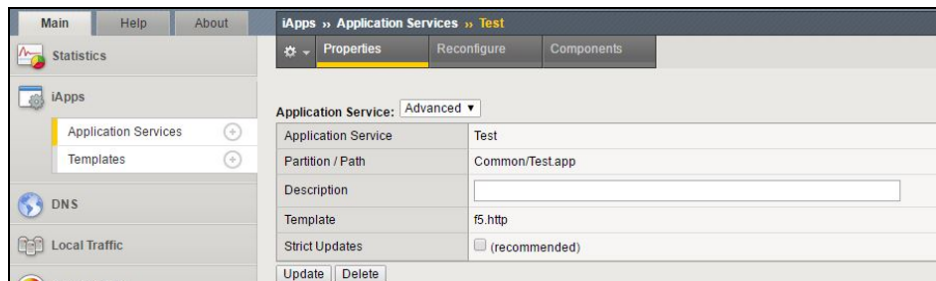


Figure 3.134: Disabling Strict Updates for an Application Service

Once this is done, proceed to associate the HTML profile with the virtual server using the procedure detailed at step 5 above.

3.4.12 Instrumenting SAP Fiori Apps

SAP Fiori is a new user experience (UX) for SAP software and applications. It provides a set of applications that are used in regular business functions like work approvals, financial apps, calculation apps and various self-service apps.

eGRUM offers support to Fiori Apps running on the SAP Cloud Platform and on the SAP On-Premises setup. The procedure to RUM-enable Fiori Apps varies with where the Apps are running - whether on the cloud or on-premise.

TheSection **3.4.12.1**topic details the procedure for instrumenting Fiori Apps running on the cloud.

TheSection **3.4.12.2**topic details the procedure for instrumenting Fiori Apps running on-premise.

3.4.12.1 Instrumenting SAP Fiori Apps Running on the SAP Cloud Platform

To RUM-enable Fiori Apps running on the SAP cloud, do the following:

1. First, manage the SAP Fiori App to be monitored as a **Real User Monitor** component using the eG admin interface (see Figure 3.135).

The screenshot shows the eG admin interface for configuring a Real User Monitor component. The top section, titled 'Component type', shows 'Real User Monitor' selected. Below this, there are fields for 'Nick name' (test), 'RUM collector' (Default Collector), and 'Remote agent' (192.168.11.102). The 'Capture UserName' option is set to 'No'. Below this, a section titled 'To instrument your html pages use the eG javascript agent' contains four download links: 'egrum.jar', 'Eg_RUM.dll', 'egrum.js', and 'Code snippet + egrum.js'. The 'Code snippet + egrum.js' link is highlighted with a red box. Below the download links, there are radio buttons for 'HTML' (selected) and 'Javascript'. At the bottom, there is a text area for the code snippet to be included in the pages, showing the following code:

```
<!-- RUM Header -->
<script charset="UTF-8" type="text/javascript"> window[egrum-start_time] = new
```

Figure 3.135: Managing a SAP Fiori APP using the eG admin interface

2. Then, to instrument the SAP Fiori App, you need to download the **egrum.js** to be embedded into the app to any folder on the local host. For that, click the **Download** button indicated by Figure 3.135.
3. Next, login to the SAP Cloud Platform (see Figure 3.136).

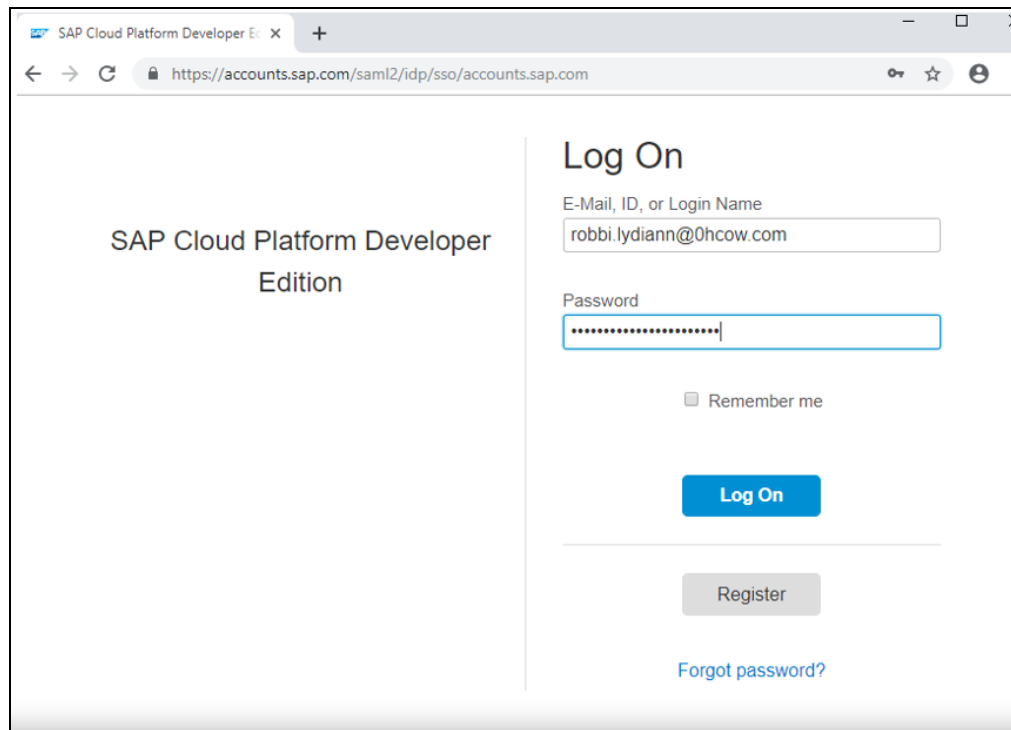


Figure 3.136: Logging into the SAP Cloud Platform

4. When Figure 3.137 appears, click on the **Services** option in the left panel of Figure 3.137. Then, with **All Categories** selected from the drop-down in the right panel, type *SAP Web IDE* in the text box adjacent to that drop-down and press **Enter** on your keyboard. This will trigger a search for the service that has the string *SAP Web IDE* as whole/part of the service name. Once **SAP Web IDE Full-Stack** appears as the search result, click on it.

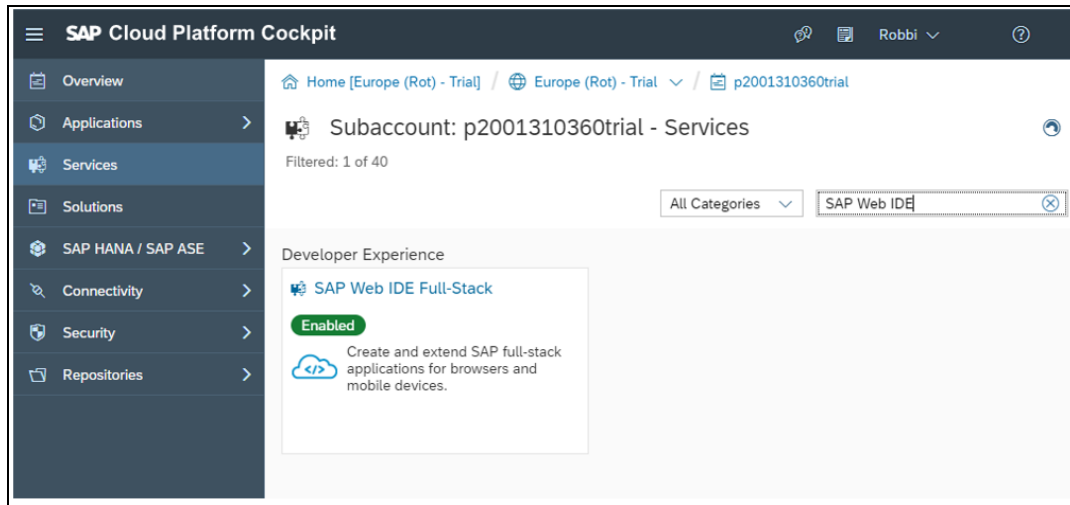


Figure 3.137: Clicking on SAP Web IDE Full-Stack

5. This will open Figure 3.138. Here, enable the SAP Web IDE Full-Stack service, if it is not done already. For that, click on the **Enabled** button in Figure 3.138. Then, click on the **Go to Service** link in Figure 3.138.

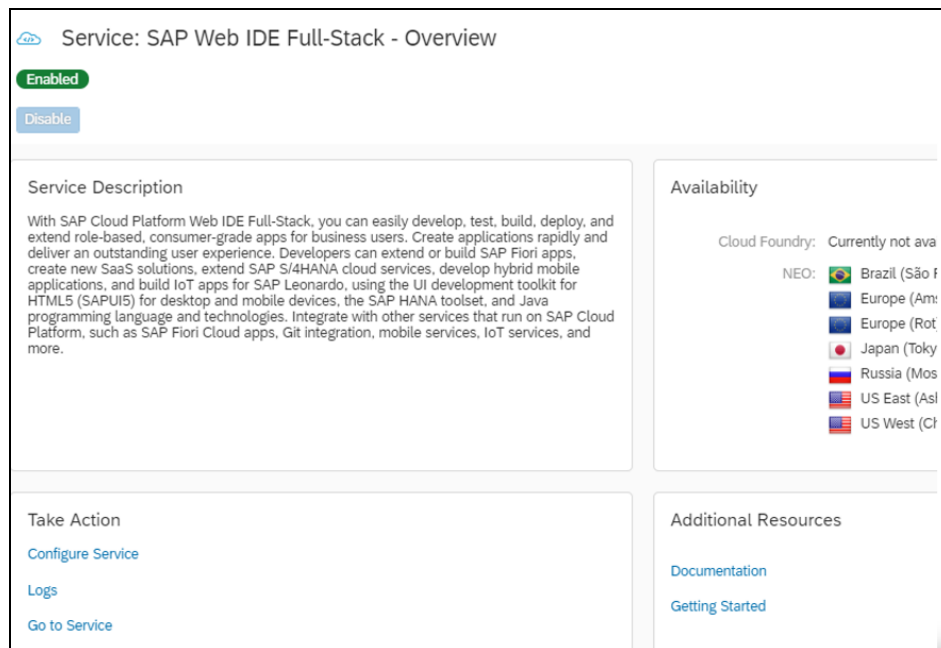


Figure 3.138: Enabling the SAP Web IDE Full-Stack service

6. Figure 3.139 then appears. Here, click on the **Gear** (⚙️) icon in the list of icons displayed in the

left panel. Then, in the right panel of Figure 3.139, turn **On** the **SAP Fiori Launchpad Extensibility** feature, and click the **Save** button to save the changes.

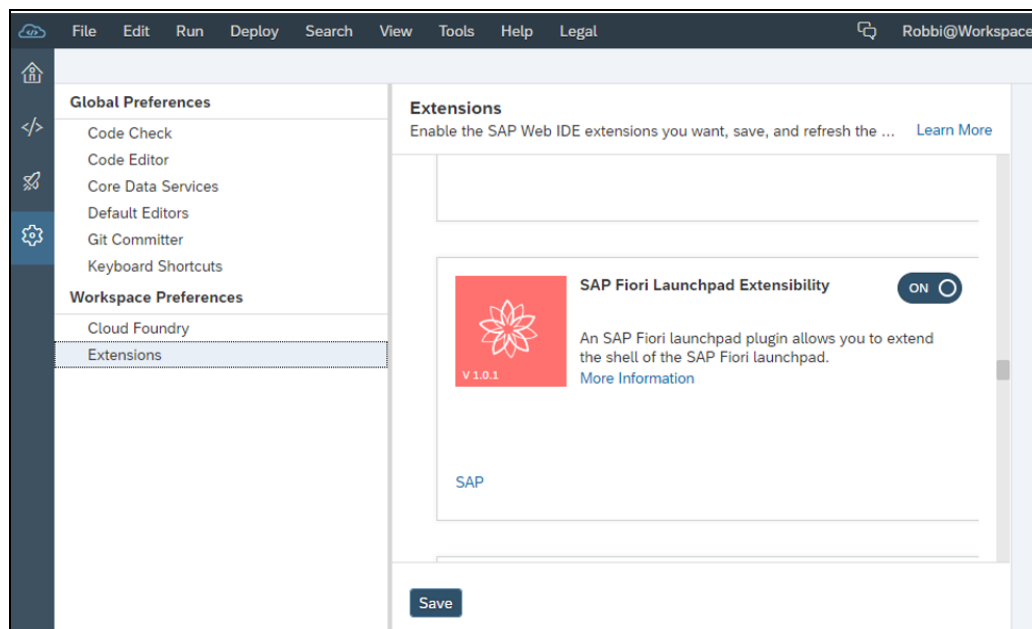


Figure 3.139: Enabling SAP Fiori Launchpad Extensibility

7. Next, click on the  icon in the list of icons displayed in the left panel of Figure 3.139. This will display the **Workspace** folder in the right panel. Then, right-click on the **Workspace** folder, move your mouse pointer over **New**, and select the **Project from Template** option (see Figure 3.140).

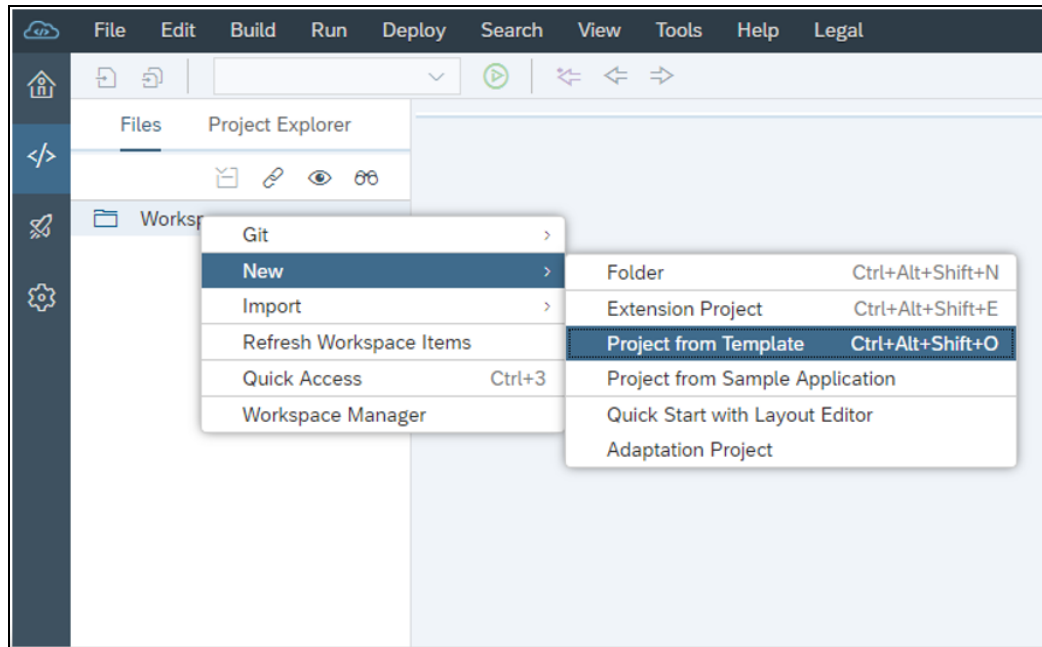


Figure 3.140: Choosing to create a new Project from a template

8. Using Figure 3.141 that then appears, you can create a new SAP Fiori Launchpad Plugin. For that, select the **SAP Fiori Launchpad Plugin** template displayed therein and click the **Next** button.

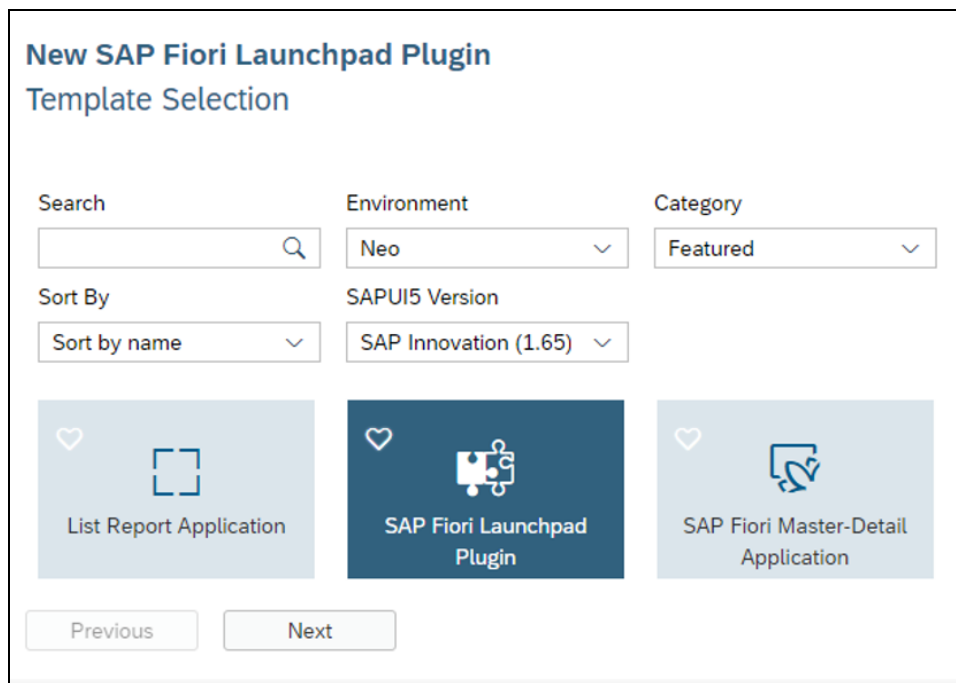


Figure 3.141: Choosing to create a new SAP Fiori Launchpad Plugin

9. Figure 3.142 will then appear. Enter the name of the new project in the **Project Name** text box in Figure 3.142. Type **eGRUMPlugin** as the name of the new project. Then, click the **Next** button to proceed.

The screenshot shows a web-based dialog titled "New SAP Fiori Launchpad Plugin" with a sub-header "Basic Information". The dialog has a breadcrumb trail at the top: "Template Selection" > "Basic Information" > "Template Customization" > "Confirmation". The "Basic Information" tab is currently active. Below the header, there is a label "Project Name*" followed by a text input field containing the text "eGRUMPlugin". At the bottom of the dialog, there are two buttons: "Previous" and "Next".

Figure 3.142: Providing the name of the new project

10. Figure 3.143 will then appear. Here, type *com.eg.rum* as the **Plugin ID**, *eG RUM Plugin* as the **Title**, and click the **Finish** button to complete plugin creation.

The screenshot shows the same "New SAP Fiori Launchpad Plugin" dialog, but now the "Template Customization" tab is active. The breadcrumb trail remains the same. Under the "General Settings" section, there are two fields: "Plugin ID*" with the value "com.eg.rum" and "Title*" with the value "eG RUM Plugin". Below this, under the "Sample Code" section, there is a checkbox labeled "Add button to launchpad header" which is currently unchecked. At the bottom, there are three buttons: "Previous", "Next", and "Finish".

Figure 3.143: Assigning an ID and title to the new plugin

11. When Figure 3.144 appears, you will find that the project you just created - i.e., **eGRUMPlugin** - has been added as a sub-folder of the **Workspace** folder. Now, proceed to create a sub-folder for the **eGRUMPlugin** folder. For that, right-click on the **eGRUMPlugin** folder, move your mouse pointer over the **New** option, and select the **Folder** option (see Figure 3.144). A **New Folder** dialog box will then appear (as shown by Figure 3.144), where you can specify the **Folder Name**. Enter *lib* as the **Folder Name** of the sub-folder, and click the **OK** button in the dialog box.

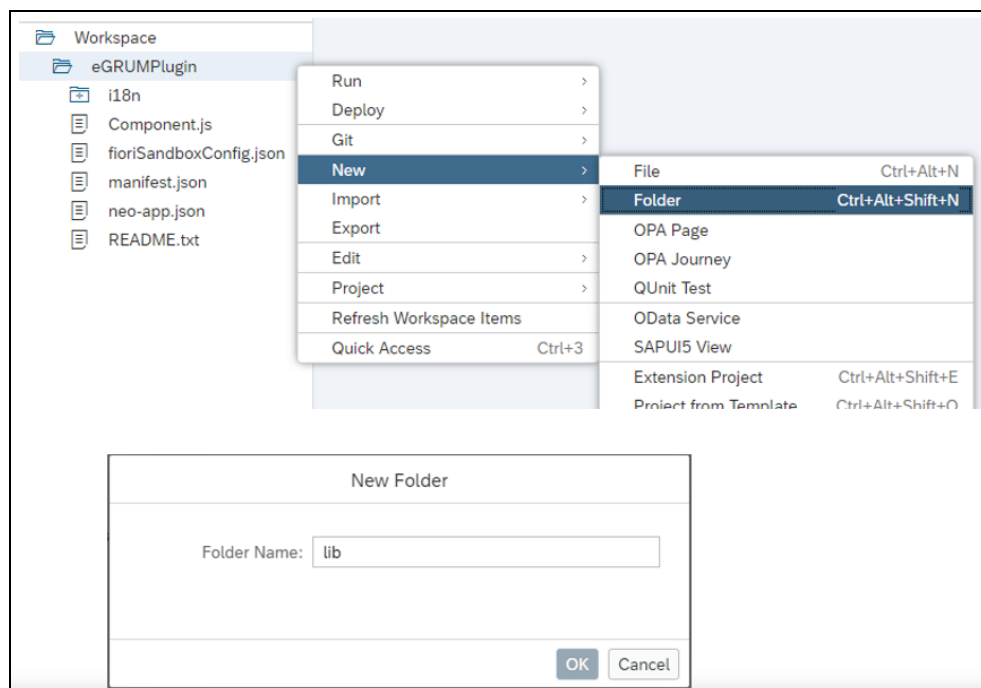


Figure 3.144: Creating a lib folder under the eGRUMPlugin folder

12. Figure 3.145 will appear, where you will be able to see the **lib** folder under the **eGRUMPlugin** folder. Now, proceed to create a file under the **lib** folder, where the RUM JavaScript code snippet needs to be stored. For that, right-click on the **lib** folder, move your mouse pointer over **New**, and choose the **File** option (see Figure 3.145). A **New File** dialog box will now appear, where you will have to specify the **File Name** of the new file. Type *egrum.js* as the name of the new file and click the **OK** button.

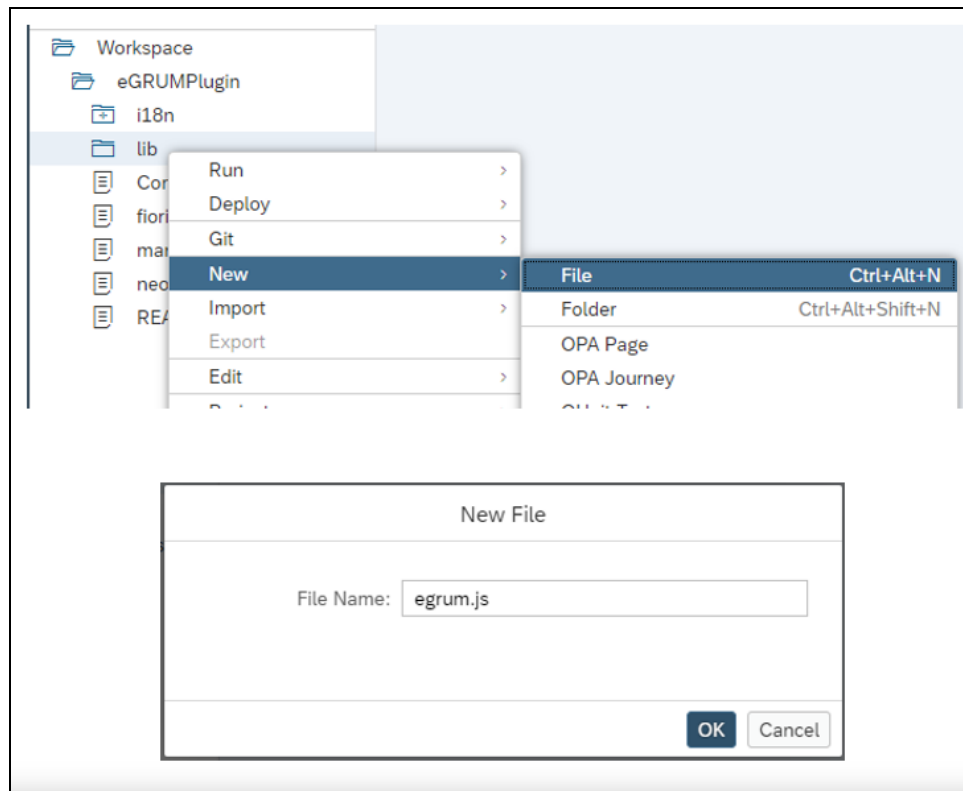


Figure 3.145: Creating the egrum.js file under the lib folder

13. Figure 3.146 then opens. You will now find that an **egrum.js** file has been created under the Workspace -> eGRUMPlugin -> lib folder. Open the **egrum.js** file by clicking on it in the left panel of Figure 3.146. An empty file will open in the right panel. Now, using a Text Editor, open the **egrum.js** file that you downloaded at step 2 above, copy the contents of that file, and paste the copied contents to the empty file in the SAP console. Press Ctrl+S on your keyboard to save these changes.



Figure 3.146: Copying the contents of the downloaded egrum.js file to the empty egrum.js file in the SAP console

14. Then, double-click the **manifest.json** file under the **eGRUMPlugin** folder to open it (see Figure 3.147). Look for the entry, **"componentName": "com.eg.rum"** in the file. Once you find the entry, append the following code block to that entry, as depicted by Figure 3.147. Then, save the changes by pressing Ctrl+S on your key board.

```
, "resources": {
  "js": [{
    "uri": "lib/egrum.js"
  }]
}
```

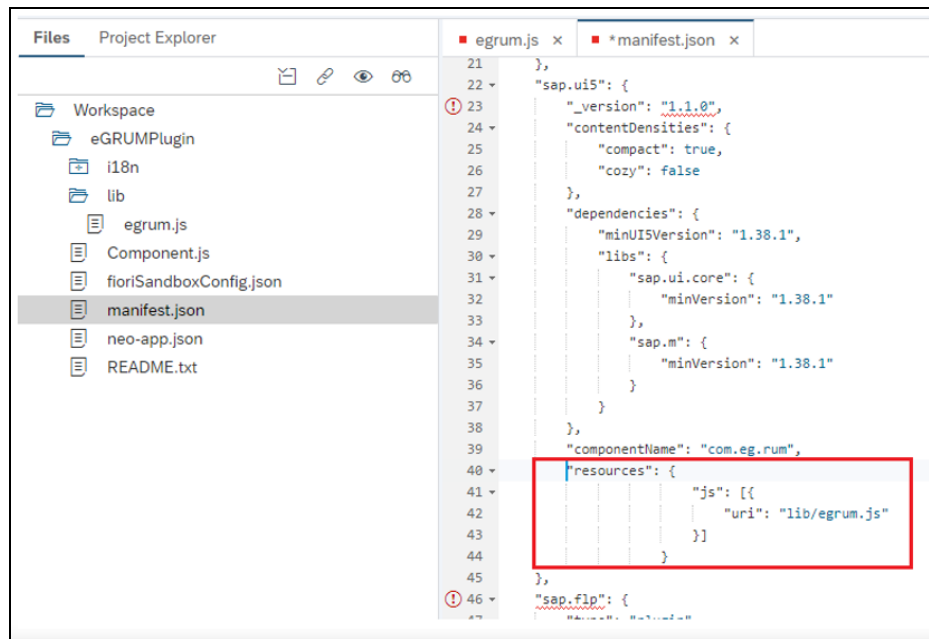


Figure 3.147: Editing the manifest.json file

15. Next, proceed to deploy the eG RUM Plugin to the SAP Cloud platform. For that, first right-click on the **eGRUMPlugin** folder, move your mouse pointer over **Deploy**, and choose the **Deploy to SAP Cloud Platform** option (see Figure 3.148).

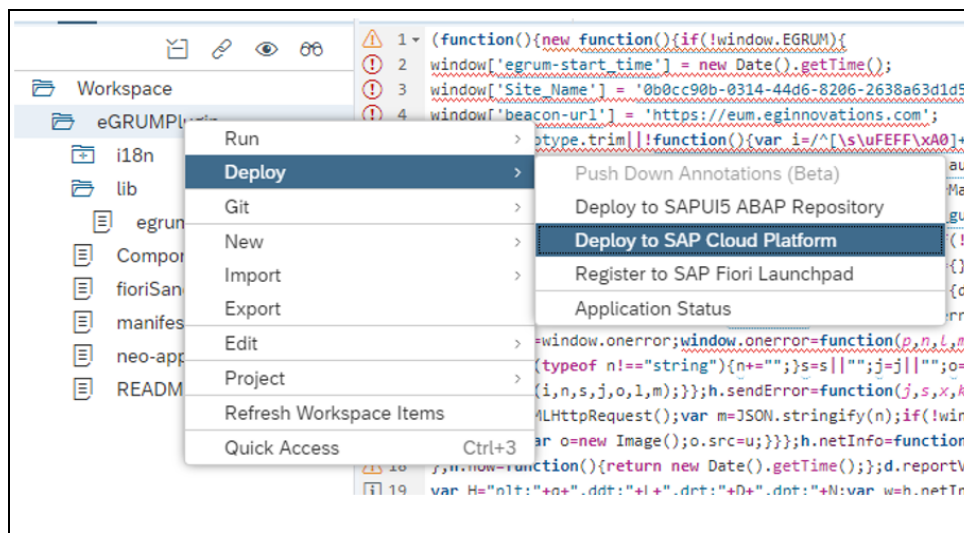


Figure 3.148: Initiating the deployment of the eGRUMPlugin to the SAP Cloud platform

16. Then, choose the **Deploy a new application** option from Figure 3.149, provide *egrumplugin* as

the **Application Name**, and click the **Deploy** button therein to deploy the plugin.

Figure 3.149: Deploying the eGRUMPlugin

17. Upon successful deployment, a message to that effect will appear (see Figure 3.150). Click the **Close** button in Figure 3.150 to close the message.

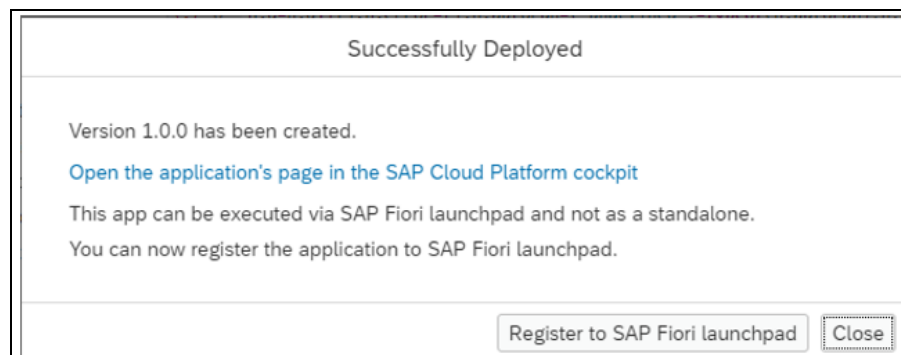


Figure 3.150: A message confirming the successful deployment of the plugin

18. Now, you need to select a portal and attach it to the eG RUM Plugin you created and deployed previously. For that, return to the **SAP Cloud Platform Cockpit** (see Figure 3.151). Click on the **Services** option in the left panel of the Cockpit. Using the Search option provided by the right panel, search for the string 'Portal'. A service named 'Portal' will then appear in the search results. Click on the **Portal** service link in Figure 3.151.

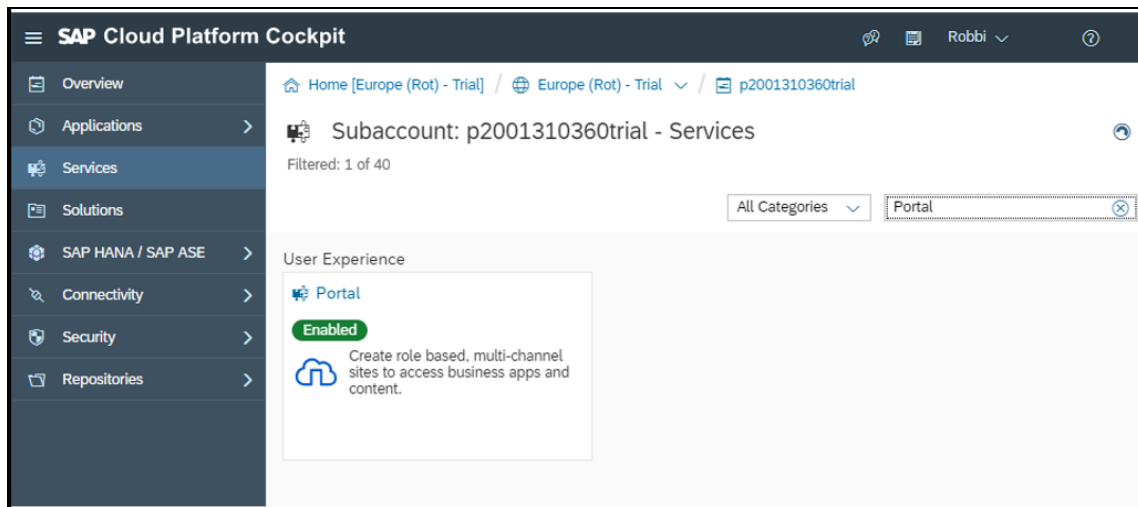


Figure 3.151: Clicking the Portal service link in the Cockpit

19. When Figure 3.152 appears, click on the Go to Service link at the bottom of Figure 3.152.

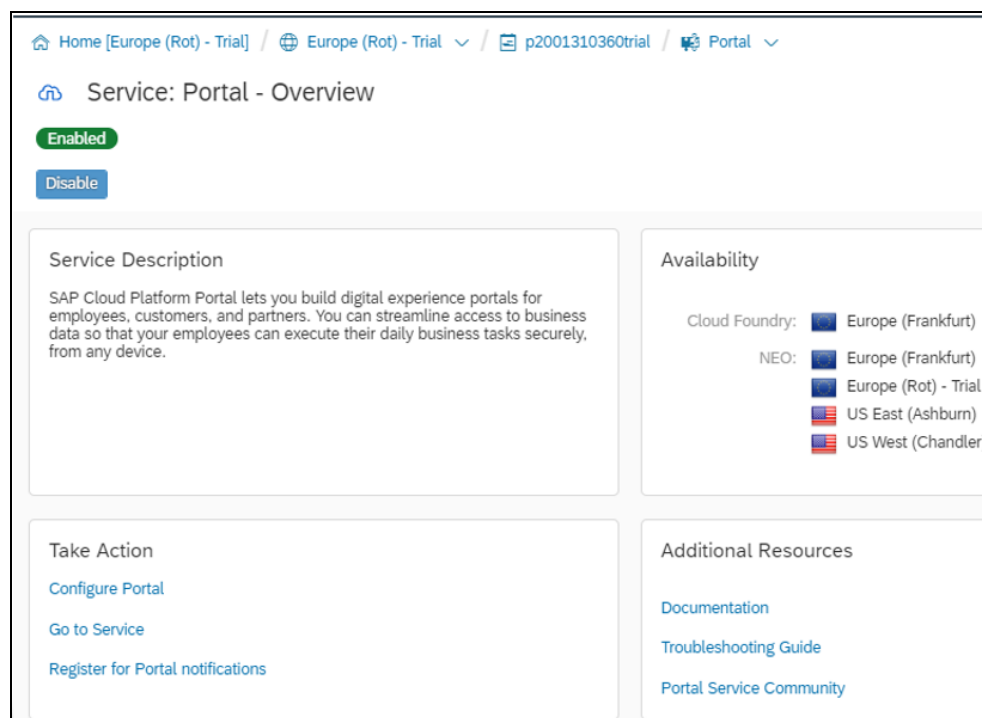


Figure 3.152: Clicking on the Go to Service link

20. Doing so will open the **Site Directory**. From here, choose the site for which you want to enable RUM. Then, click on the **Edit** link that appears when you hover your mouse pointer over

the site URL (see Figure 3.153).

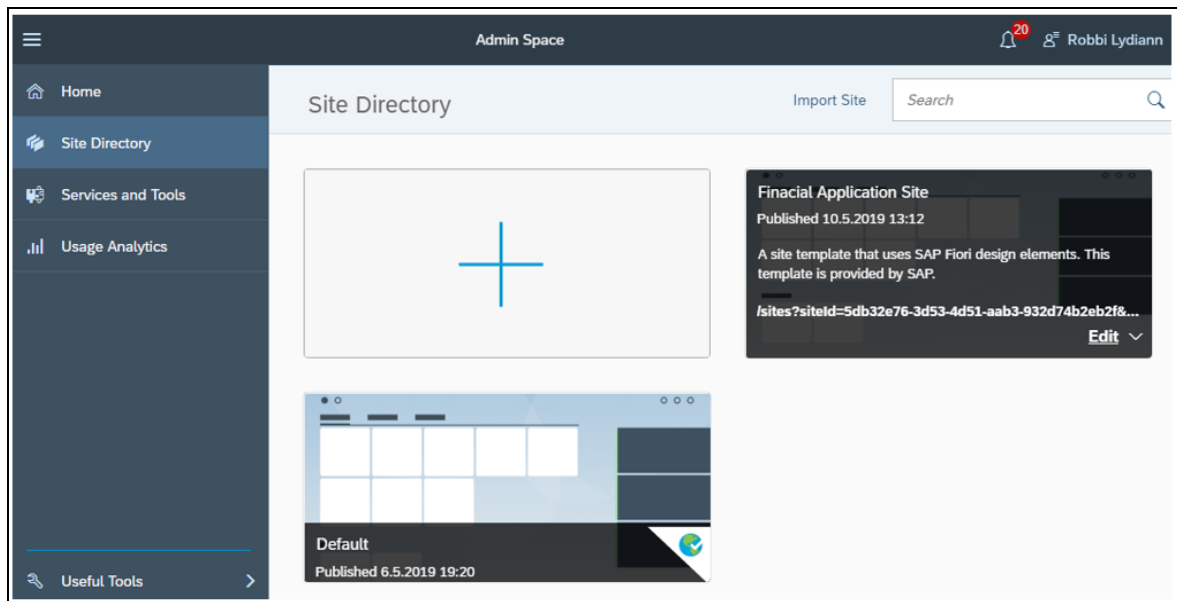


Figure 3.153: Choosing the site to be RUM-enabled

21. Figure 3.154 will then appear. Expand the **Content Management** node in the left panel of Figure 3.154, and choose the **Apps** sub-node within. Then, to add the eG RUM Plugin to the chosen site, click the  button that is last in an array of buttons that you will find at the bottom of the right panel.

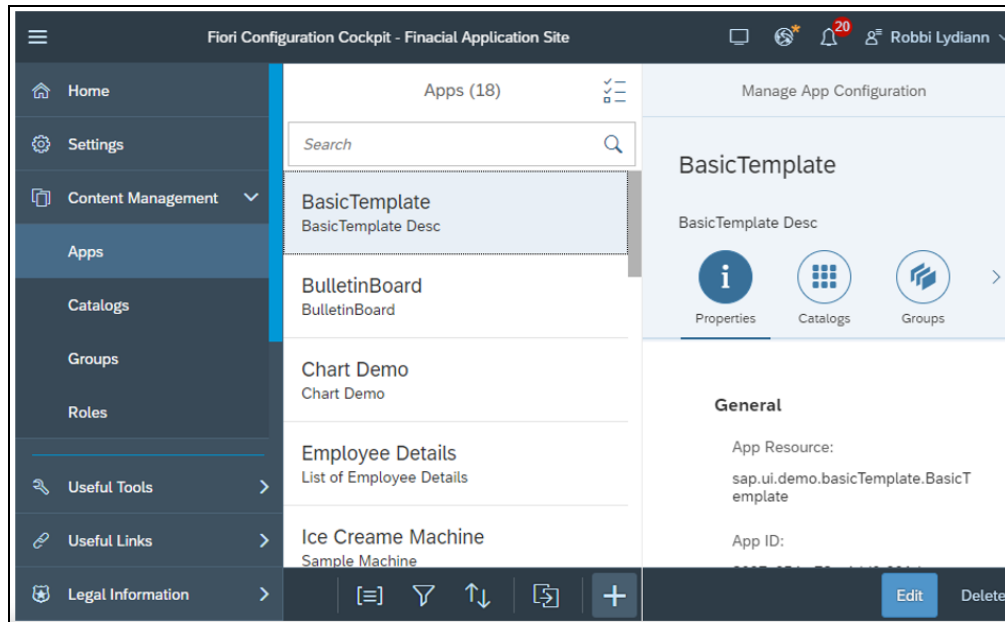


Figure 3.154: Clicking the '+' icon to add the eGRUMPlugin to the chosen site

22. Figure 3.155 will then appear. Here, you need to select the **eG RUM Plugin** as the **App Resource** you want to associate with the site. To perform this selection, you first need to browse the list of available App resources. For that, click the **Browse** (🔗) button at the end of the **App Resource** text box in Figure 3.155.

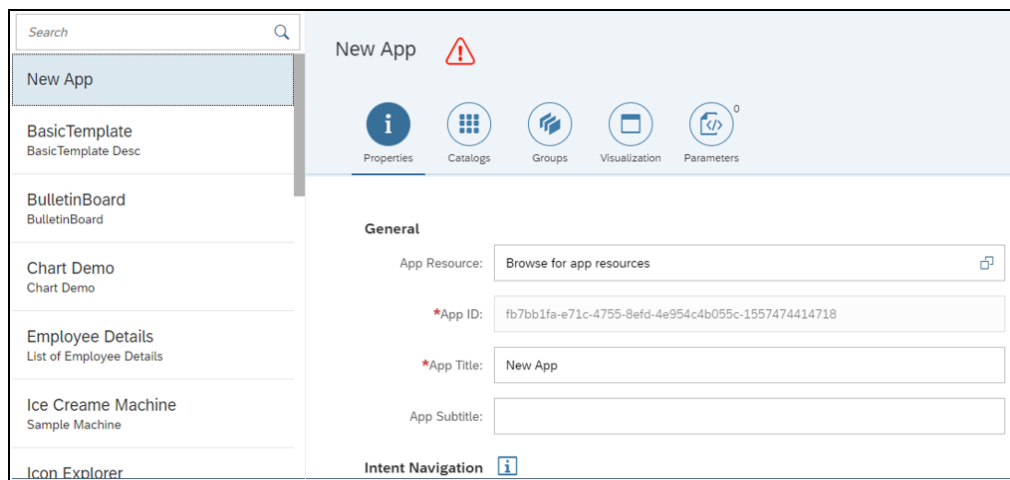


Figure 3.155: Clicking the App Resource Browse button to look for the eGRUMPlugin

23. Figure 3.156 will then appear, where the available app resources will be listed. Browse the list and select the **egrumpugin** resource from it. Then, click the **OK** button in Figure 3.156.

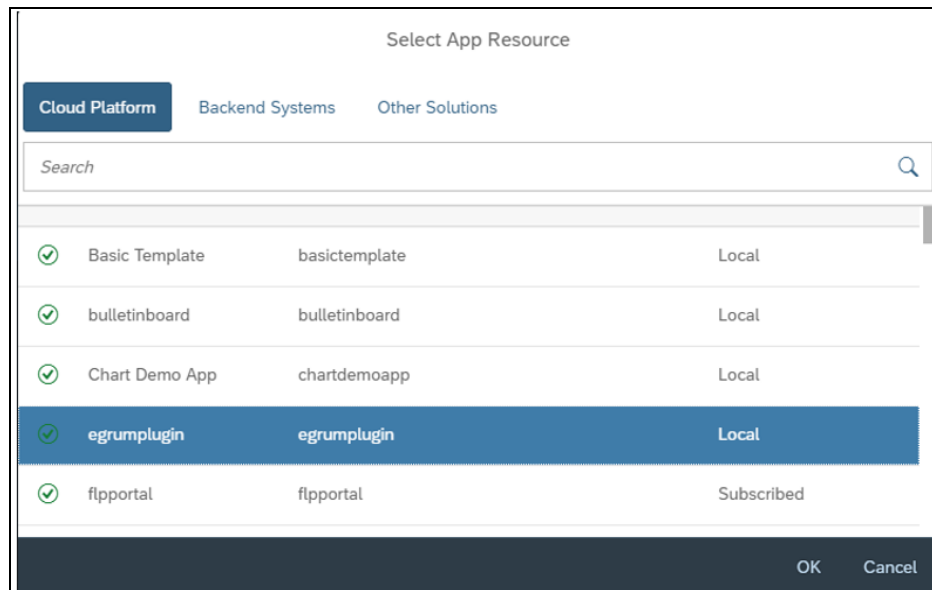


Figure 3.156: Selecting the eGRUMPlugin from the list of App Resources

24. This will bring you back to Figure 3.155. This time however, you will see that the *egrumplugin* is displayed as the **App Resource** (see Figure 3.157). Now, indicate what type of an app the *egrumplugin* is. For that, select the *Shell Plugin* option from the **App Type** drop-down (see Figure 3.155). Then, click the **Save** button.

The screenshot shows a configuration window for an SAPUI5 application. The 'App Resource' field is set to 'egrumpugin'. The 'App ID' field is set to 'SAPUI5'. The 'App Title' field is set to 'URL'. The 'App Subtitle' field is set to 'SAPUI5 News'. The 'Intent Navigation' section is expanded, showing a list of options: 'SAP GUI', 'Web Dynpro ABAP', 'SAP Mobile Documents', 'Dynamic URL', and 'Business Server Page (BSP)'. The 'App Type' dropdown menu is open, showing 'SAPUI5' as the selected option. The 'Save' and 'Cancel' buttons are visible at the bottom right.

Field	Value
App Resource	egrumpugin
*App ID	SAPUI5
*App Title	URL
App Subtitle	SAPUI5 News
Intent Navigation	SAP GUI, Web Dynpro ABAP, SAP Mobile Documents, Dynamic URL, Business Server Page (BSP)
App Type	SAPUI5

Figure 3.157: Setting Shell Plugin as the App Type

25. When Figure 3.158 appears, click on **Catalogs** to assign the eG RUM Plugin to a catalog. Then, click the **Edit** button in Figure 3.158.

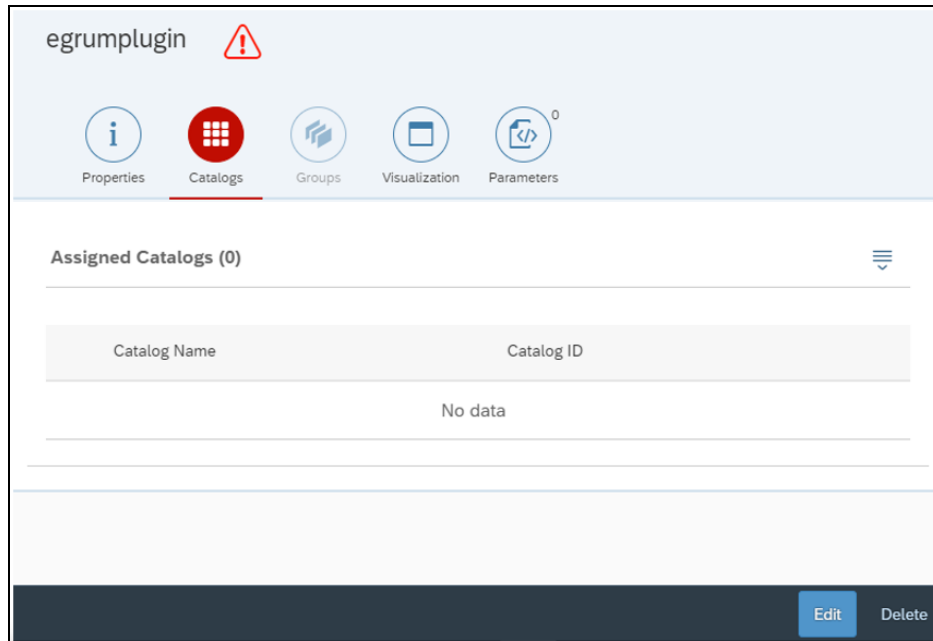


Figure 3.158: Clicking on the Catalogs icon

26. Figure 3.159 then appears. Click on the '+' icon in Figure 3.159 to add a catalog.

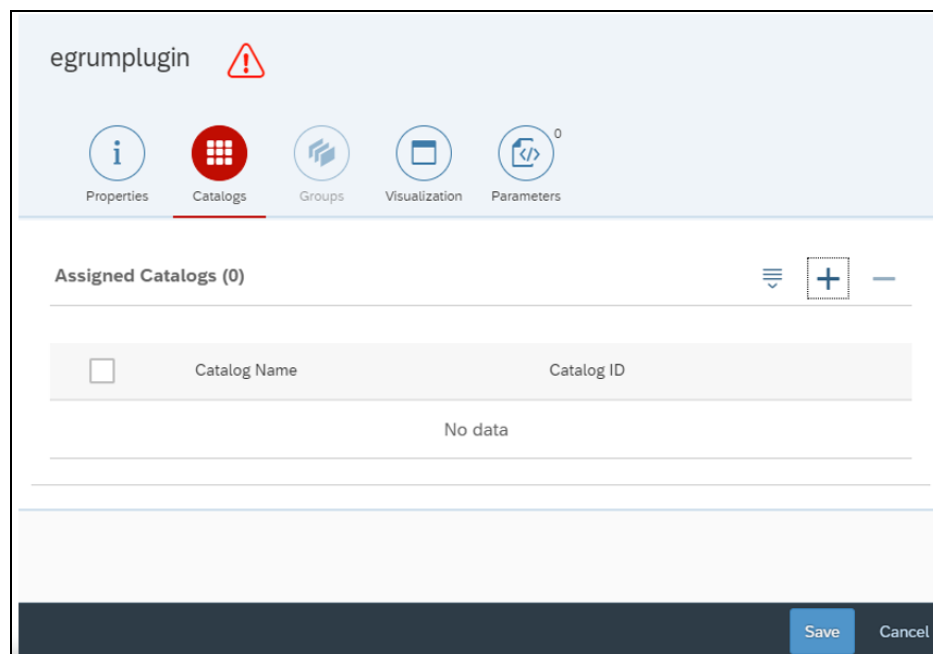


Figure 3.159: Clicking on the '+' icon to add a catalog

27. From the list of catalogs that appears, select the catalog in which you want to add the

eG RUM Plugin (see Figure 3.160). Then, click the **OK** button.

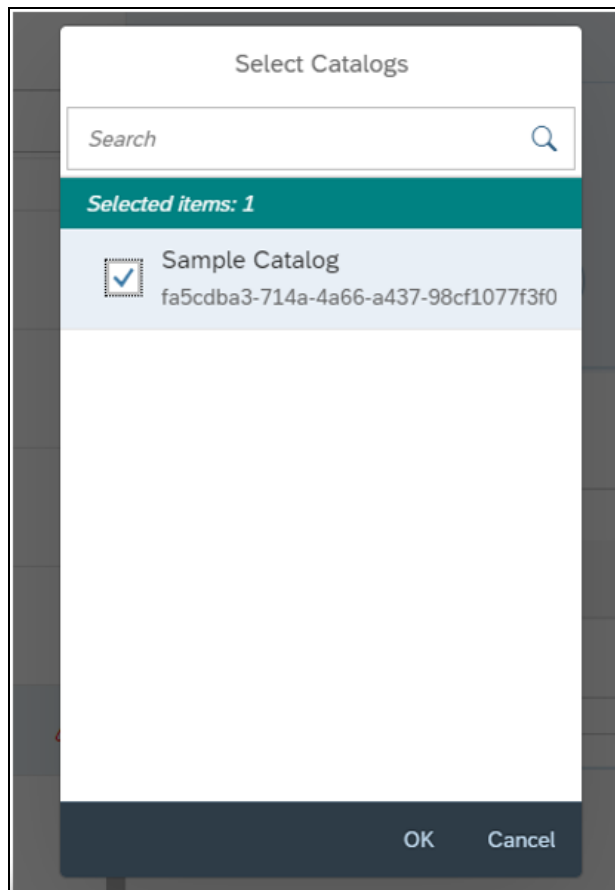


Figure 3.160: Selecting the catalog in which the eG RUM Plugin is to be added

28. Upon selection, you will return to Figure 3.159. This time however, you will see the catalog you selected displayed under **Assigned Catalogs** (see Figure 3.161). Now, click the **Save** button to save the changes.

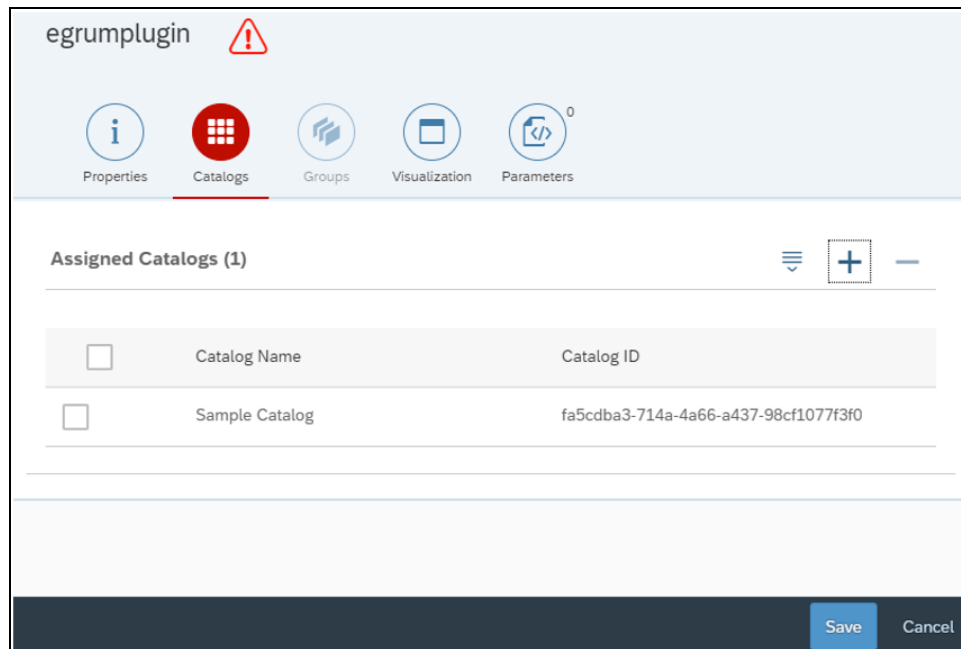


Figure 3.161: The selected catalog displayed under Assigned Catalogs

29. Finally, to make sure that all changes are applied, you will have to publish the site. For that, when Figure 3.162 appears, click on the  button at the right, top corner of Figure 3.162. A **Publish Site** dialog box will then appear. Check the **Clear HTML5 application cache** check box in the dialog box, and then click the **Publish and Open** button therein, to publish the site.

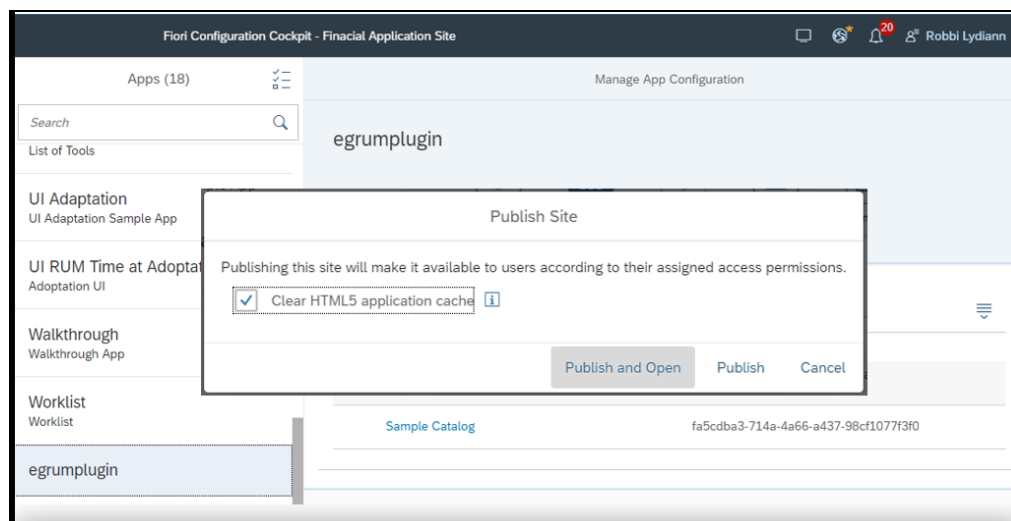


Figure 3.162: Publishing the site

Once the site is successfully published, it automatically opens. You can now invoke Developer Tools from the site. If you find a GET request to the site, as indicated by Figure 3.163, you can conclude

that the `egrum.js` has been successfully loaded. If you find a POST request bearing the RUM collector's IP and port number (as indicated by Figure 3.163), it means that the browser is sending beacons to the eG RUM collector.

200	GET	flpportal-p2001313505trial.dispatc...	manifest.json?ts=201904240804	155.56.219.2...	xhr	json
200	GET	flpportal-p2001313505trial.dispatc...	manifest.json?ts=201904240804	155.56.219.2...	xhr	json
200	GET	flpportal-p2001313505trial.dispatc...	style.css	155.56.219.2...	stylesheet	css
200	GET	flpportal-p2001313505trial.dispatc...	i18n_en.properties?ts=201904240804	155.56.219.2...	xhr	plain
200	GET	flpportal-p2001313505trial.dispatc...	i18n.properties?ts=201904240804	155.56.219.2...	xhr	plain
200	GET	flpportal-p2001313505trial.dispatc...	egrum.js?ts=201904240804	155.56.219.2...	xhr	js
200	POST	192.168.8.38:7088	rdr	192.168.8.38:...	xhr	gif
200	GET	flpportal-p2001313505trial.dispatc...	i18n.properties?ts=201904240804	155.56.219.2...	xhr	plain
200	POST	192.168.8.38:7088	rdr	192.168.8.38:...	xhr	gif
200	POST	192.168.8.38:7088	rdr	192.168.8.38:...	xhr	gif
200	POST	192.168.8.38:7088	rdr	192.168.8.38:...	xhr	gif
200	GET	sapui5.hana.ondemand.com	Component-preload.js	23.9.76.39:443	script	js

Figure 3.163: Verifying successful loading of `egrum.js` and the transmission of beacons to collector using Developer tools

3.4.12.2 Instrumenting SAP Fiori Apps Running On-Premise

The broad steps for instrumenting SAP Fiori Apps running on-premise are as follows:

1. Configure the SAP Cloud Connector; this step is required only if you do not have on-premise full stack SAP Web IDE;
2. Configure the SAP Cloud Platform destination;
3. Configure SAP full stack Web IDE and deploy it to on-premise application;
4. Configure SAP user and backend;
5. Verify whether configuration works

These steps have been elaborately discussed below:

1. First, manage the SAP Fiori Apps to to be monitored as a **Real User Monitor** component using the eG admin interface (see Figure 3.135).
2. Then, to instrument the SAP Fiori Apps, you need to download the **egrum.js** to be embedded into the app to any folder on the local host. For that, click the **Download** button indicated by Figure 3.135.
3. Then, connect to the SAP Cloud Connector using the URL, `https://localhost:8443/`, and login to it

using valid login credentials (see Figure 3.164). Before attempting to connect, make sure that the SAP Cloud Connector is installed.

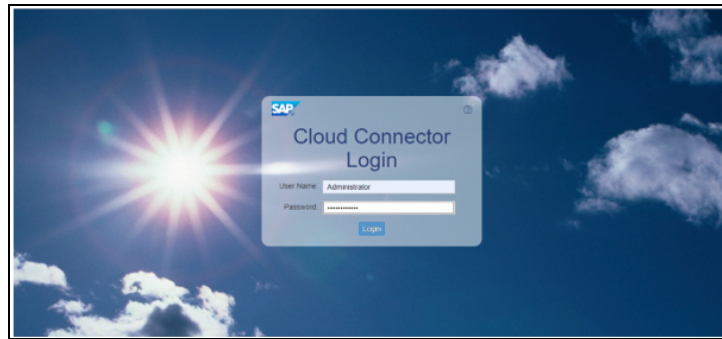


Figure 3.164: Logging into the SAP Cloud Connector

- After logging in, proceed to create a sub-account. For that, in the SAP Cloud Platform Cockpit that appears, click on the option that corresponds to your SAP cloud platform subscription - eg., Cloud Foundry Trial, Neo Trial etc. In the case of the example illustrated by Figure 3.165, the **Neo Trial** option is clicked as the Neo trial subscription is in use.

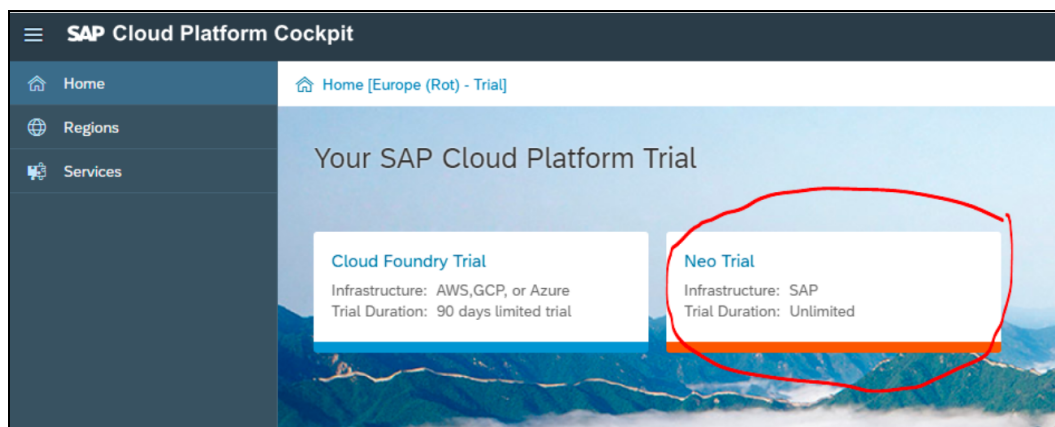


Figure 3.165: Clicking on the option that corresponds to the SAP Cloud Platform subscription in use

- In the page that appears next, click on the **Create Subaccount** button to create a new subaccount. Figure 3.166 will then appear.

Figure 3.166: Creating a new subaccount

6. In Figure 3.166, select a **Region** for the new subaccount. Then, specify the **Subaccount** name. To know what the subaccount name is, first minimize the **Add Subaccount** dialog box of Figure 3.166, so that the SAP Cloud Platform Cockpit comes into view (see Figure 3.167). Scroll down the cockpit until you can view a **Subaccount Information** section. The value displayed under **Name** in this section is the name of the new subaccount you are creating. Copy this name and paste it against **Subaccount** name in Figure 3.166. Then, proceed to specify the **Subaccount User** name. This should be the same as the **Subaccount** name, but sans the 'trial' suffix. For instance, if the **Subaccount** name is *p1940914553trial*, the **Subaccount User** name should be just *p1940914553*. Then, specify the **Password** of the user. Leave the **Location ID** empty and click the **Save** button in Figure 3.166 to save the new subaccount.

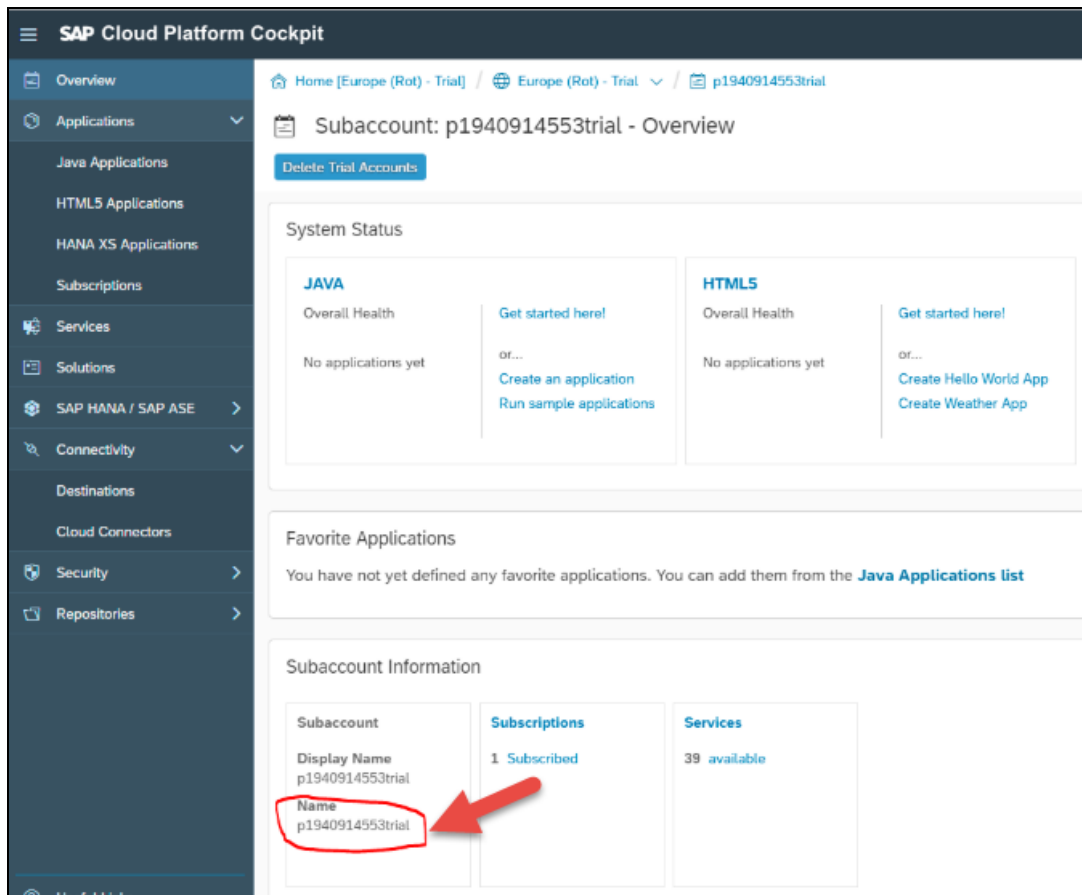


Figure 3.167: Determining the Subaccount name

- Next, proceed to create a virtual system on the Cloud Platform, which will map to the system On-premise. For that, first select the **Cloud to On-Premise** option in the left panel of the SAP Cloud Platform Cockpit (see Figure 3.168).

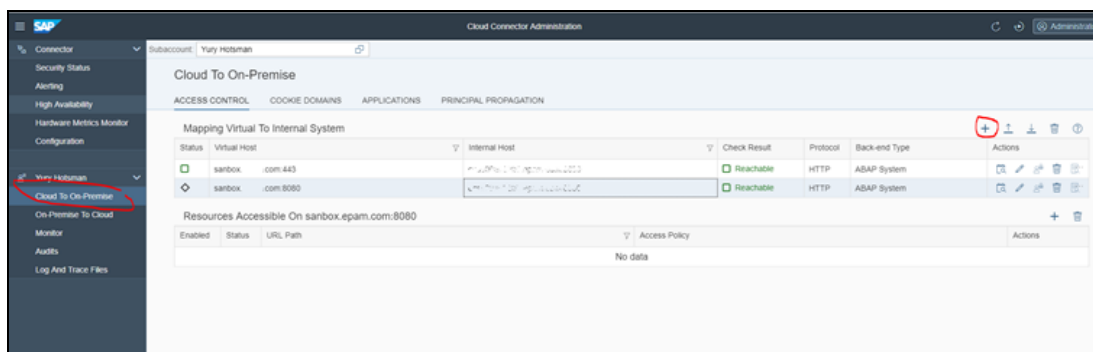
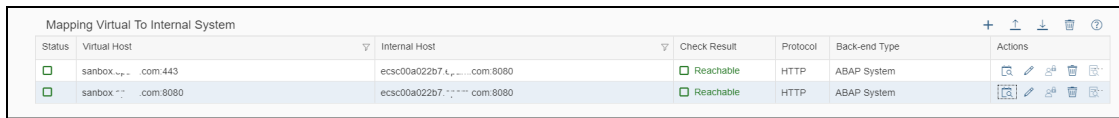


Figure 3.168: Clicking the Cloud to On-Premise option

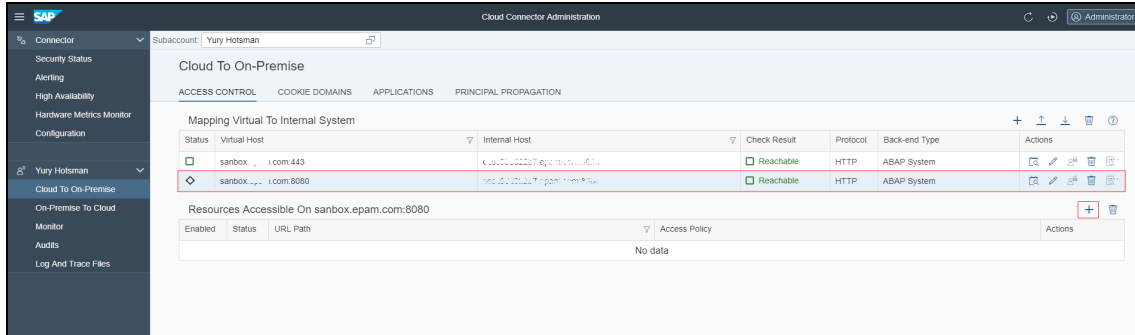
8. When the **Cloud To On-Premise** section appears in the right panel, click the '+' button (indicated by Figure 3.168) in that section to map the virtual system to the internal system. In the dialog box that appears next (not shown in this topic), set the **Back-end** type as *ABAP System* and set **Protocol** as *HTTP*. Then, specify the **Internal Host** name. To know the internal host name, go to the Application Server section in your SAP Logon. Then, configure the **Internal Port**. To know the internal port, execute the transaction SMICM, and go to Services -> HTTP -> Port. Then, provide any name against **Virtual Host** and any port number against **Virtual Port**. Let the **Principal Type** be *None*. Once you **Save** this configuration, the new mapping will appear in the **Mapping Virtual to Internal System** list, as depicted by Figure 3.169.



Status	Virtual Host	Internal Host	Check Result	Protocol	Back-end Type	Actions
☐	sanbox-...com:443	ecsc00a022b7...com:8080	☒ Reachable	HTTP	ABAP System	[Icons]
☐	sanbox-...com:8080	ecsc00a022b7...com:8080	☒ Reachable	HTTP	ABAP System	[Icons]

Figure 3.169: The new mapping appended to the Mapping Virtual To Internal System list

9. If you now scroll down the **Cloud to On-Premise** section, you will find a **Resources Accessible** section. Select the new mapping you created from the **Mapping Virtual to Internal System** section of Figure 3.168, and then click the '+' button in the **Resources Accessible** section to add a new resource (see Figure 3.170).

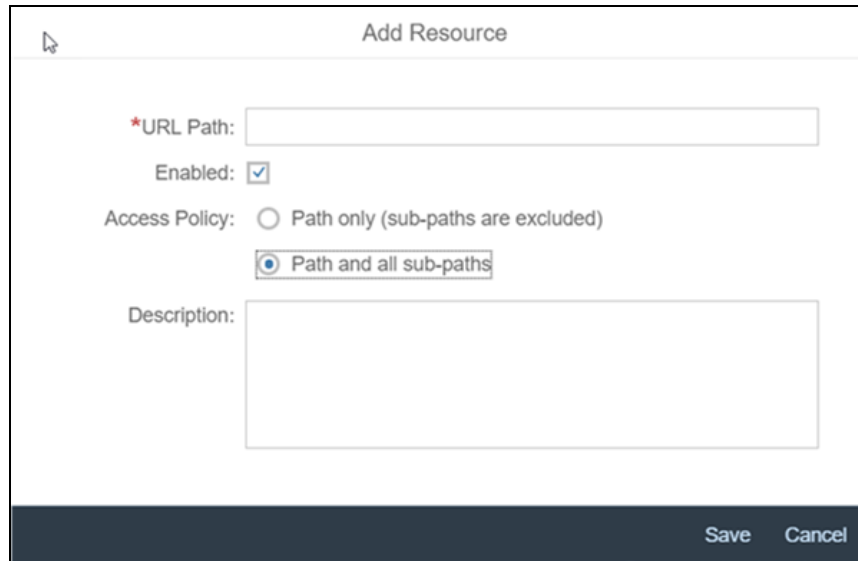


Status	Virtual Host	Internal Host	Check Result	Protocol	Back-end Type	Actions
☐	sanbox-...com:443	ecsc00a022b7...com:8080	☒ Reachable	HTTP	ABAP System	[Icons]
☒	sanbox-...com:8080	ecsc00a022b7...com:8080	☒ Reachable	HTTP	ABAP System	[Icons]

Enabled	Status	URL Path	Access Policy	Actions
No data				

Figure 3.170: Selecting the new mapping and clicking the '+' icon to add a new resource

10. Figure 3.171 will then appear. To RUM-enable SAP Fiori Apps, we need to add the eG RUM Plugin as a cloud application. To make the necessary resources available to this cloud application, you first need to add each resource the eG RUM Plugin needs to the SAP Cloud Platform (see Figure 3.171). The first resource to be added is the URL to which the eG RUM Plugin should be deployed. Therefore, first enter **sap/bc/ui5_ui5** in the **URL Path** text box in Figure 3.171. Then, set *Path and all sub-paths* as the **Access Policy**, and click the **Save** button to save the changes.

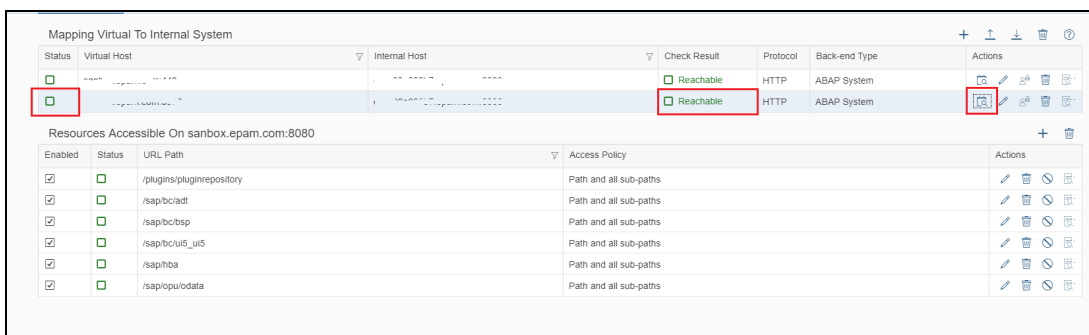


The 'Add Resource' dialog box contains the following fields and options:

- *URL Path:** A text input field.
- Enabled:** A checked checkbox.
- Access Policy:** Two radio button options:
 - ☐ Path only (sub-paths are excluded)
 - ☒ Path and all sub-paths
- Description:** A large text area.
- Buttons:** 'Save' and 'Cancel' buttons at the bottom right.

Figure 3.171: Adding a new resource

11. You will now return to Figure 3.170, where the new resource will be appended to the list of resources in the **Resources Accessible** section. Next, proceed to add another resource that the eG RUM Plugin needs. For that, click the '+' icon in the **Resources Accessible** section of Figure 3.170. When Figure 3.171 comes up again, set the **URL Path** as `/sap/opu/odata`, set **Path and all sub-paths** as the **Access Policy**, and click the **Save** button to save the changes. This path allows the eG RUM Plugin access to odata gateway services it requires for its functioning.
12. Once both the resources are added, proceed to check whether the cloud to on-premise connection is working. For that, select the new mapping from the **Mapping Virtual to Internal System** section, and click the **Check Availability** button corresponding to it. If the **Check Result** column returns the value 'Reachable', you can conclude that the mapping is working.



The 'Mapping Virtual To Internal System' table shows two mappings. The second mapping is highlighted with a red box, and its 'Check Result' is 'Reachable'.

Status	Virtual Host	Internal Host	Check Result	Protocol	Back-end Type	Actions
☒	...	...	Reachable	HTTP	ABAP System	[Icons]
☒	...	...	Reachable	HTTP	ABAP System	[Icons]

Resources Accessible On sanbox.epam.com:8080

Enabled	Status	URL Path	Access Policy	Actions
☒	☒	/plugins/pluginrepository	Path and all sub-paths	[Icons]
☒	☒	/sap/bc/adt	Path and all sub-paths	[Icons]
☒	☒	/sap/bc/bsp	Path and all sub-paths	[Icons]
☒	☒	/sap/bc/ui5_ui5	Path and all sub-paths	[Icons]
☒	☒	/sap/hba	Path and all sub-paths	[Icons]
☒	☒	/sap/opu/odata	Path and all sub-paths	[Icons]

Figure 3.172: Checking connectivity

- Next, proceed to configure a destination. For that, once again login to the SAP Cloud Platform and select your cloud subscription. Figure 3.173 will then appear. In the left panel of Figure 3.173, expand the **Connectivity** node, and select the **Destinations** node within.

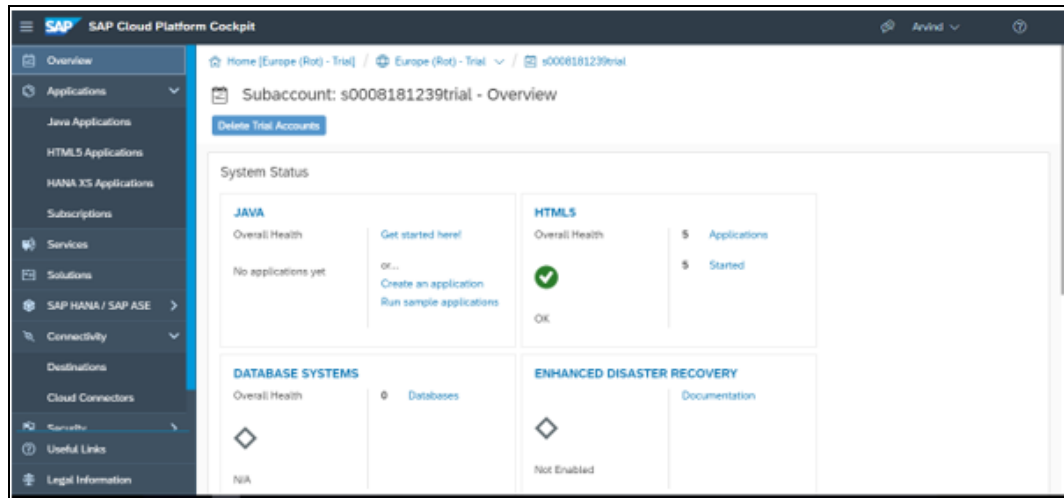


Figure 3.173: Selecting the Destinations option from the Connectivity node

- This will invoke Figure 3.174. Here, click the **New Destination** button to create a new destination.

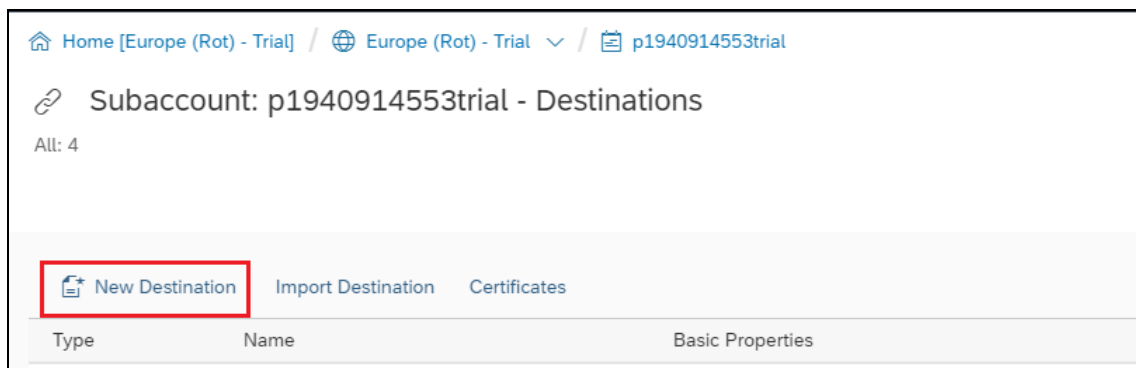


Figure 3.174: Clicking the New Destination button

- Figure 3.175 will then appear.

The image shows a 'Destination Configuration' form. On the left, there are fields for: *Name (with a dropdown), Type (set to HTTP), Description, Location ID (highlighted with a red box), *URL (with a dropdown), Proxy Type (set to OnPremise), Authentication (set to BasicAuthentication), User (set to yhotsman), and Password (masked with asterisks). On the right, under 'Additional Properties', there are fields for: sap-client (set to 100), WebIDEEnabled (set to true), WebIDESystem (set to ...), and WebIDEUsage (set to odata_gen,odata_abap,c). A 'New Property' button is at the top right.

Figure 3.175: Configuring a new destination

16. In Figure 3.175, first assign any unique **Name** to the new destination. Then, set **Type** to *HTTP*. Provide any **Description** for the new destination. Next, in the **URL** text box, provide a URL of this format: `<Name_of_virtual_host_in_new_mapping_you_created>:<Port_number_of_virtual_host>`. For the name and port number of virtual host, refer to step 9 above. Then, set **ProxyType** to *OnPremise* and **Authentication** to *BasicAut*. Against **User** and **Password**, specify your system username and password. Then, under **Additional Properties**, set the **WebIDEEnabled** flag to *true*. Specify the name of the backend system in the **WebIDESystem** text box, and enter *odata_gen,odata_abap,dev_abap* in the **WebIDEUsage** text box. Once all the destination parameters are configured, click **Save** to create the new destination.
17. After saving the new destination, click the **Check Connection** button (indicated by Figure 3.176) corresponding to the new destination to check if the destination is accessible. If it is, a message to that effect will pop up, as depicted by Figure 3.176.

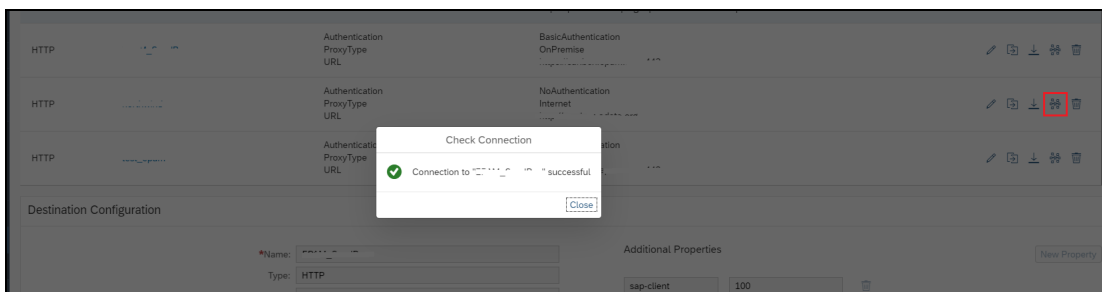


Figure 3.176: Checking connectivity to the new destination

18. Next, proceed to configure the SAP full stack Web IDE in the SAP Cloud Platform. For that, open the SAP Cloud Platform using the link, <https://cockpit.hanatrial.ondemand.com/cockpit/#/home/trial>.
19. When Figure 3.177 appears, click **Launch SAP Web IDE**.

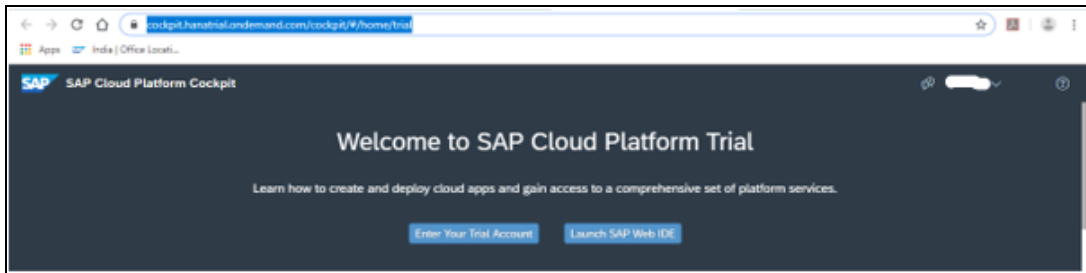


Figure 3.177: Launching SAP Web IDE

20. Figure 3.178 will then appear. First, enable SAP Fiori Launchpad extensibility for Sap Web IDE. For that, click the **Preferences** icon in the left panel of Figure 3.178, select **Extensions**, and turn **On** the **SAP Fiori Launchpad Extensibility** flag (see Figure 3.178).

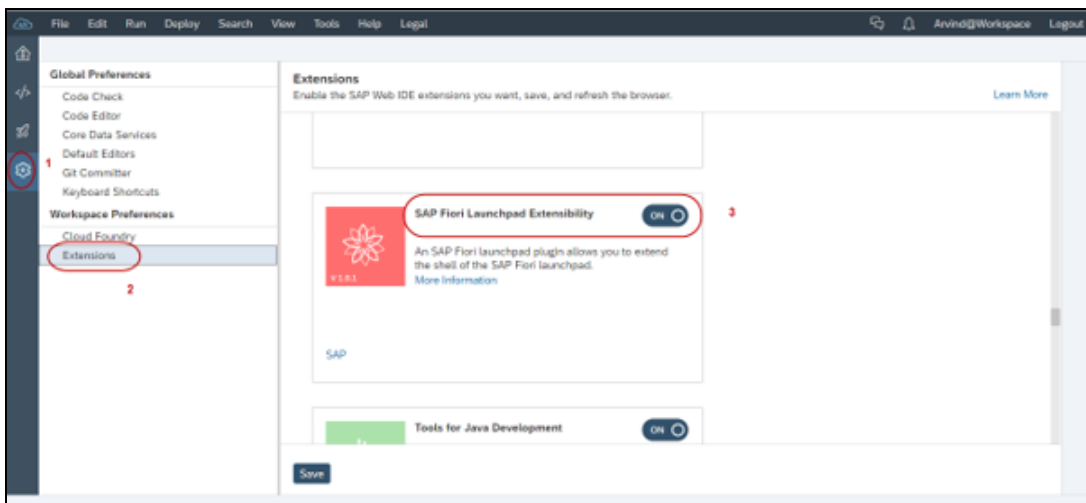


Figure 3.178: Enabling SAP Fiori Launchpad Extensibility

21. Next, proceed to create a new project from template. For that, follow the menu sequence File -> New -> Project from Template (see Figure 3.179).

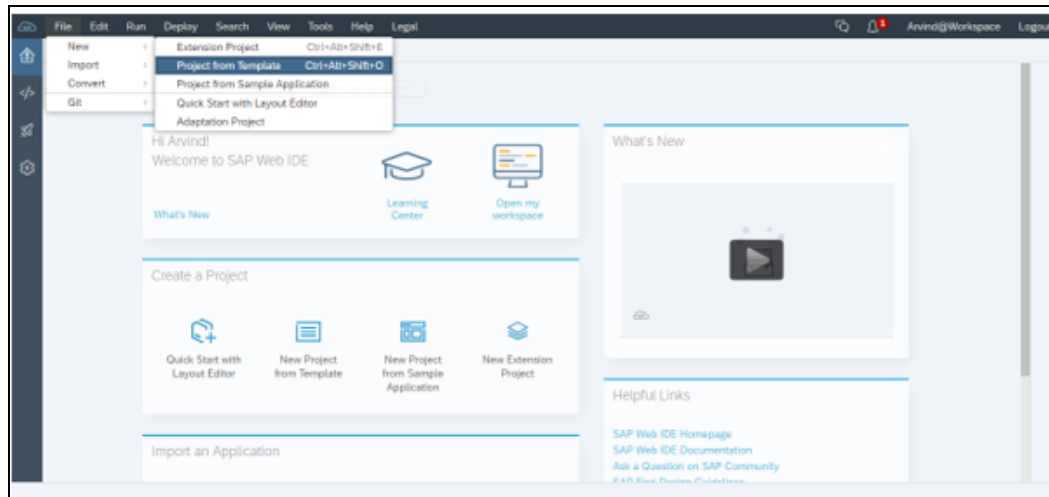


Figure 3.179: Selecting the option to create a new project from template

22. When Figure 3.180 appears, click on **SAP Fiori Launchpad Plugin**.

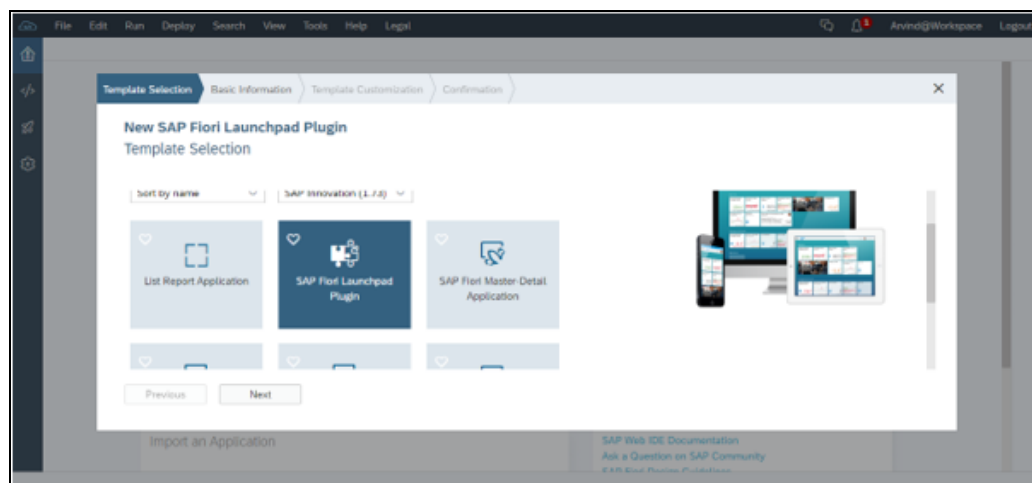
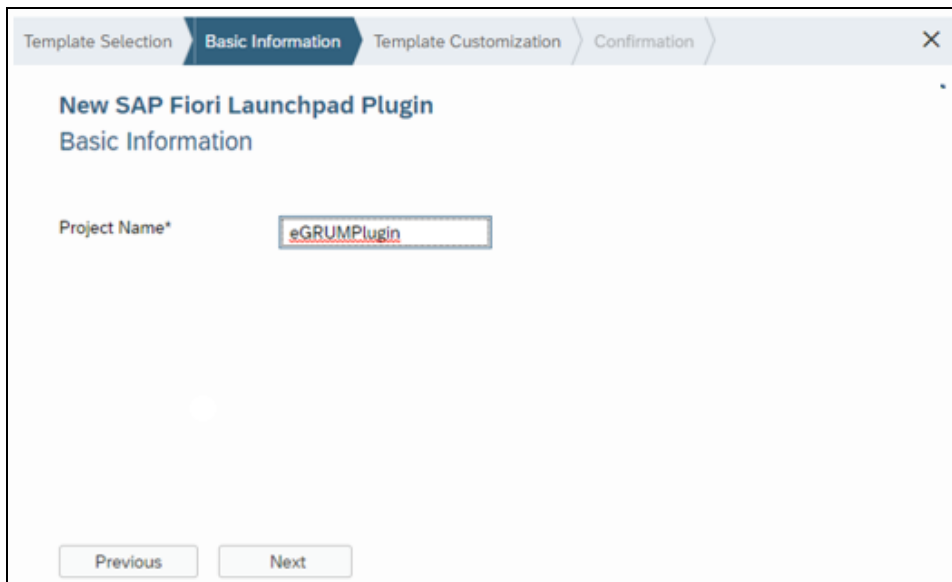


Figure 3.180: Selecting the SAP Fiori Launchpad Plugin

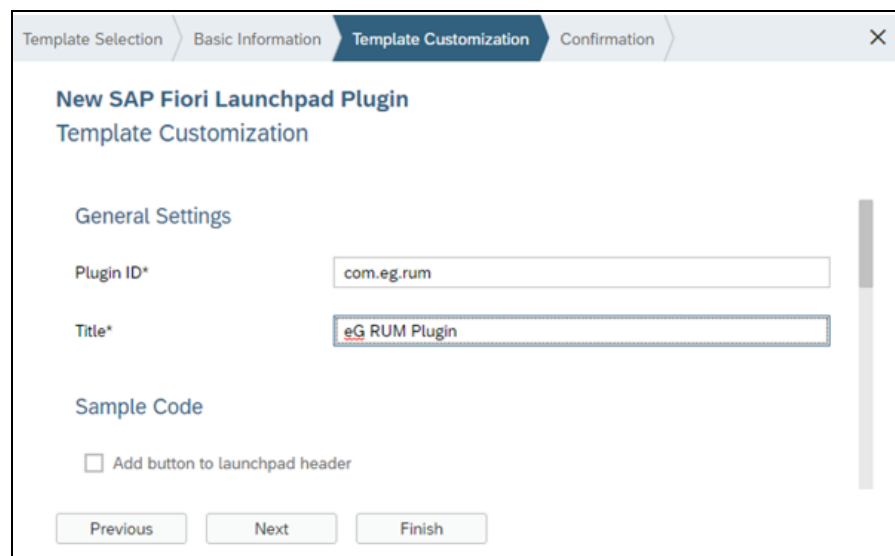
23. When prompted for a **Project Name**, specify *eGRUMPlugin* (see Figure 3.181). Then, click the **Next** button to proceed.



The screenshot shows a web-based dialog titled 'New SAP Fiori Launchpad Plugin' with a sub-header 'Basic Information'. The dialog has four tabs: 'Template Selection', 'Basic Information' (active), 'Template Customization', and 'Confirmation'. A 'Project Name*' field is present with the value 'eGRUMPlugin' entered. At the bottom, there are 'Previous' and 'Next' buttons.

Figure 3.181: Specifying the new project name

24. In Figure 3.182 that appears next, specify *com.eg.rum* as the **Plugin ID** and *eG RUM Plugin* as the **Title**. Then, click the **Finish** button to complete the plugin creation process.



The screenshot shows the same dialog, but now on the 'Template Customization' tab. Under the 'General Settings' section, the 'Plugin ID*' field contains 'com.eg.rum' and the 'Title*' field contains 'eG RUM Plugin'. Below this, under 'Sample Code', there is a checkbox labeled 'Add button to launchpad header' which is currently unchecked. At the bottom, there are 'Previous', 'Next', and 'Finish' buttons.

Figure 3.182: Providing the plugin ID and title

25. You will now find that an **eGRUMPlugin** folder is created under the **Workspace** folder (see Figure 3.183). Now, create a **lib** folder under **eGRUMPlugin**. For that, right-click on **eGRUMPlugin**, move your mouse pointer over **New**, and select the **Folder** option.

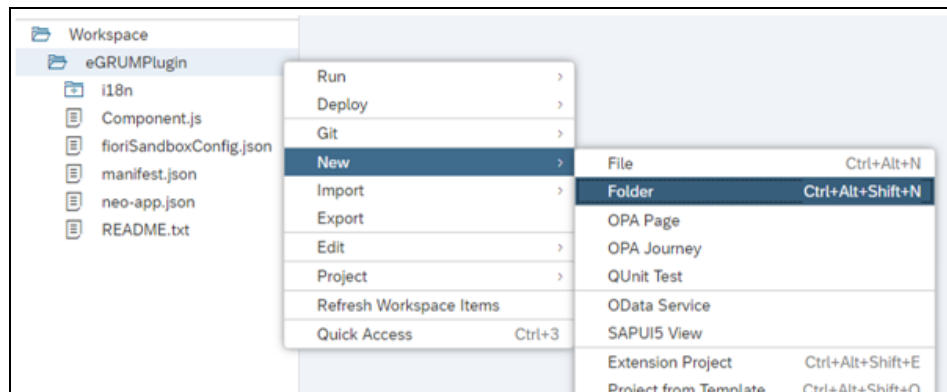


Figure 3.183: Creating a sub-folder under eGRUMPlugin

26. When Figure 3.184 appears, specify *lib* as the **Folder Name**.



Figure 3.184: Naming the sub-folder 'lib'

27. Next, create an **egrum.js** file under the **lib** folder. For that, right-click on the **lib** folder, move your mouse pointer over **New**, and select the **File** option. When Figure 3.186 appears, specify *egrum.js* against **File Name**.

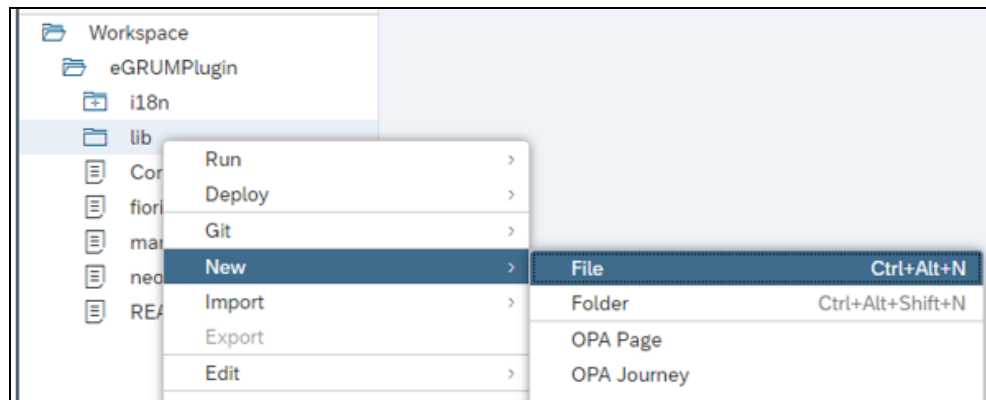


Figure 3.185: Creating a file under the lib folder

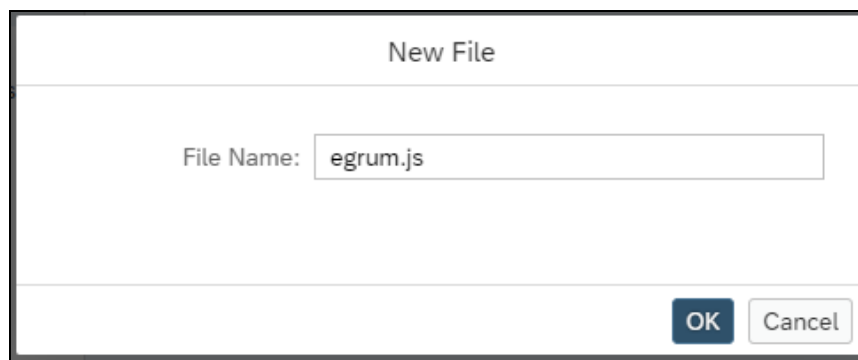


Figure 3.186: Specifying the new file name as egrum.js

28. You will now find that an **egrum.js** file has been created under the Workspace -> eGRUMPlugin -> lib folder. Open the **egrum.js** file by clicking on it in the left panel of Figure 3.187. An empty file will open in the right panel. Now, using a Text Editor, open the **egrum.js** file that you downloaded at step 2 above, copy the contents of that file, and paste the copied contents to the empty file in the SAP Cloud Platform cockpit. Press Ctrl+S on your keyboard to save these changes (see Figure 3.187).

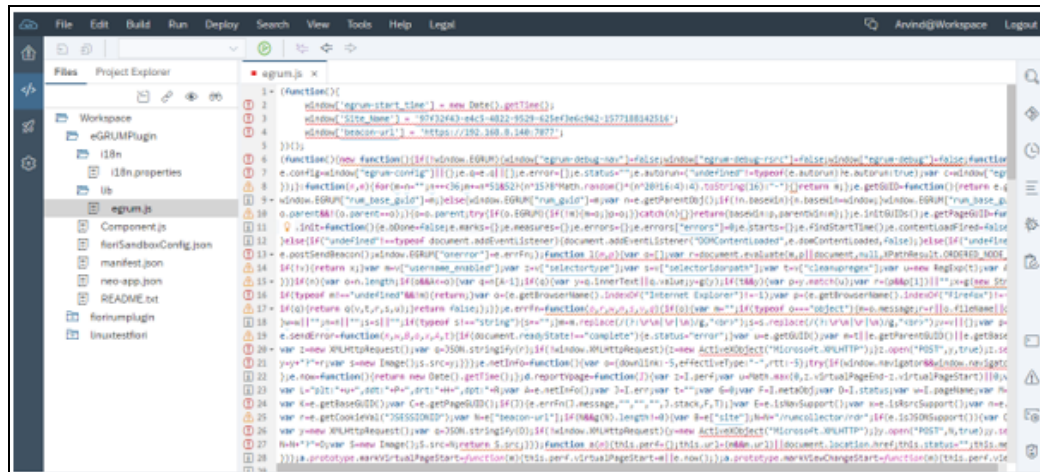


Figure 3.187: Pasting the contents of the downloaded egrum.js file into the empty egrum.js file

29. Then, double-click the **manifest.json** file under the **eGRUMPlugin** folder to open it (see Figure 3.188). Look for the entry, **"componentName": "com.eg.rum"** in the file. Once you find the entry, append the following code block to that entry, as depicted by Figure 3.147. Then, save the changes by pressing Ctrl+S on your key board.

```
, "resources": {
  "js": [{
    "uri": "lib/egrum.js"
  }]
}
```

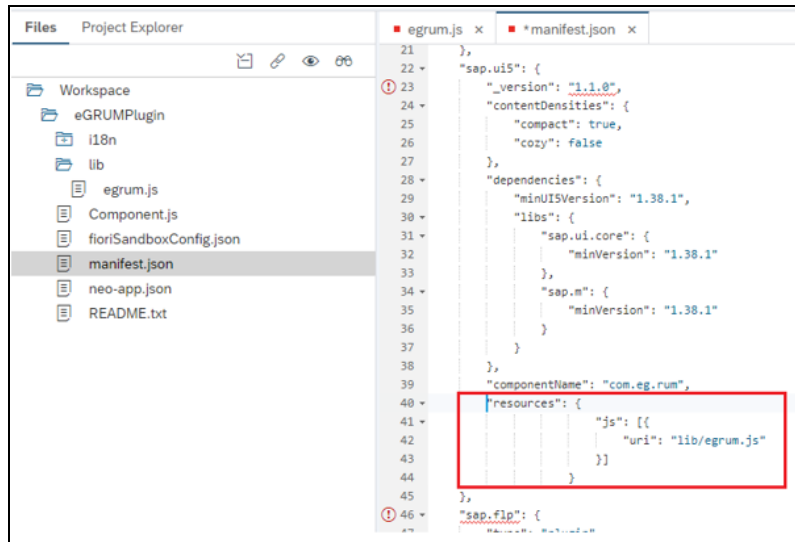


Figure 3.188: Editing the manifest.json file when RUM-enabling SAP Fiori Apps On-Premise

30. Now, deploy the plugin to the SAP ABAP repository. For that, right-click on **eGRUMPlugin**, move your mouse pointer over **Deploy**, and select **Deploy to SAPUI5 ABAP Repository** (see Figure 3.189).

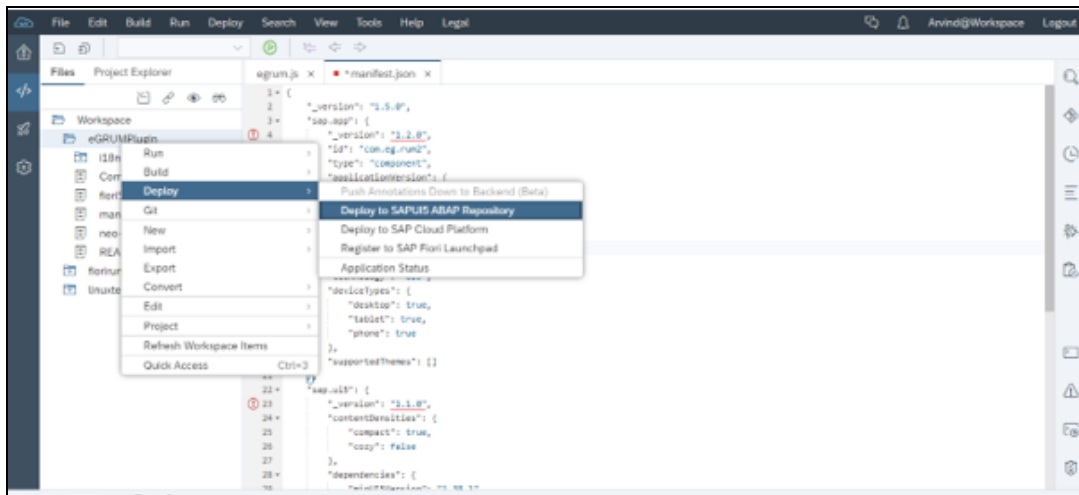


Figure 3.189: Deploying the eGRUMPlugin to the SAPUI5 ABAP Repository

31. A **Deployment Options** tab page will then appear. From the **System** drop-down here, select the destination to which the eG RUM Plugin is to be deployed. Here, make sure you select the same destination you created using steps 14-17 above. Then, select the **Deploy a new application** option to deploy the plugin as a new application. Click the **Next** button to proceed.

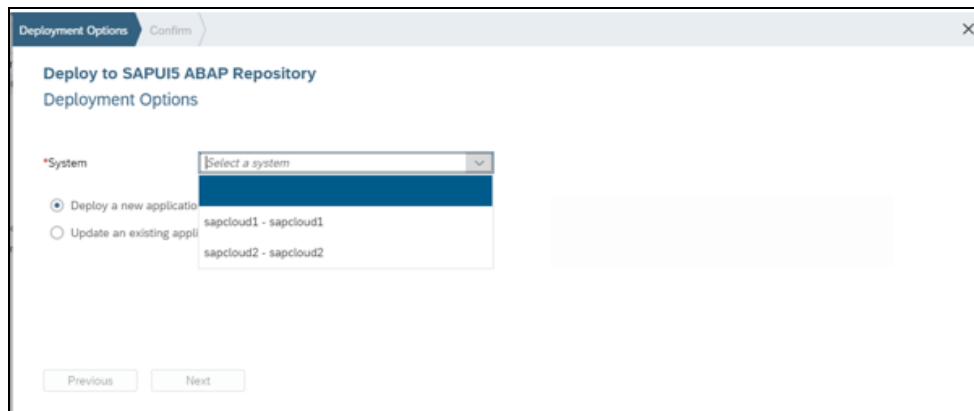


Figure 3.190: Selecting the deployment destination

32. In Figure 3.191, specify the **Name** of the new application, provide a **Description** for the application, and indicate the **Package** name. In the case of the example illustrated by Figure 3.191, the application **Name** is *zegrumfiori*. Now, click **Next** to move to the next step.

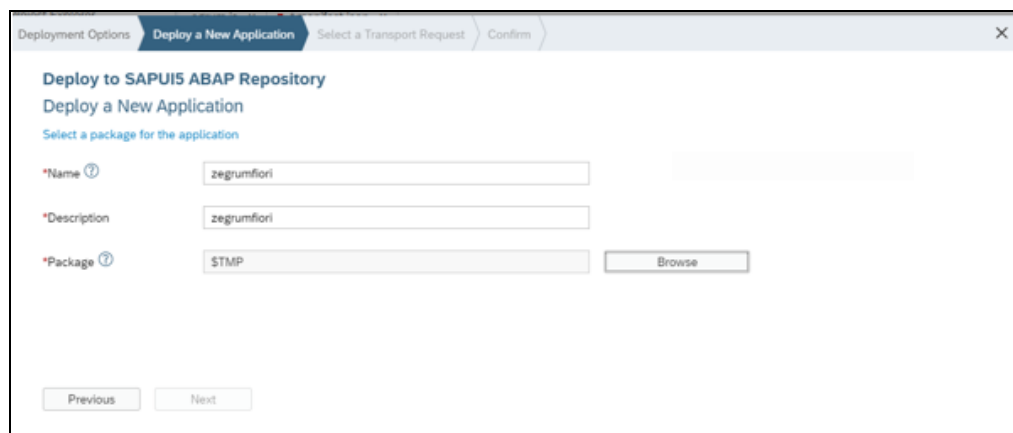


Figure 3.191: Deploying a new application

33. When Figure 3.192 appears, click **Finish** to save the changes and complete the deployment process.

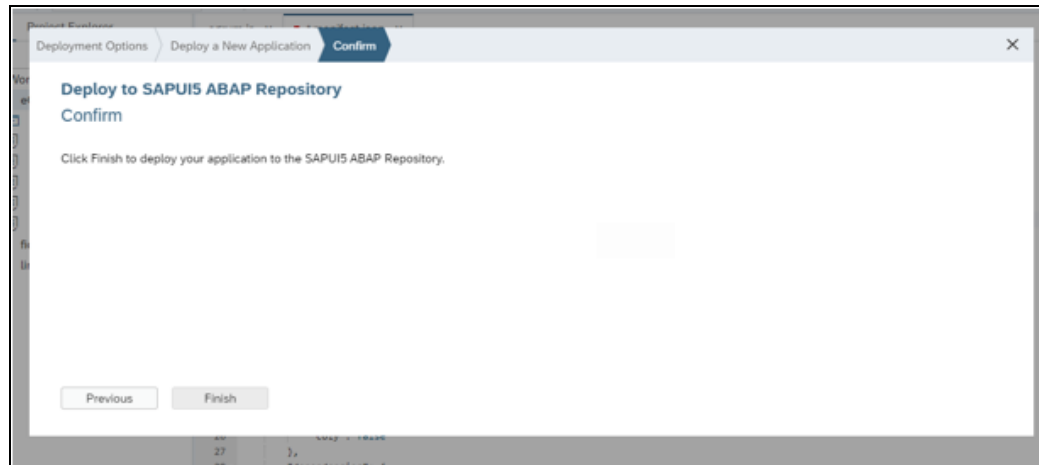


Figure 3.192: Finishing the application deployment

34. Once application deployment is successful, a service with the same name as the application will be automatically created in the SAP Fiori system. In the case of our example therefore, a service named *zegrumpfiori* will be automatically created.
35. Let us now proceed to activate this service in the SAP backend. For that, first login to the SAP GUI as a SAP administrator. In the SAP GUI, type *SICF* in the text box you will find at the right top corner of the GUI, and press the Enter key on the keyboard. Figure 3.193 will then appear. Here, to search for the service that was automatically created, type the name of the service against **ServiceName**. Once the service is found, the tree-structure under **Virtual Hosts/Services** will change to indicate the location of the service. You have to activate only that service that is listed under *ui5_ui5 -> sap* in the tree-structure. For that, right-click on the service name and select the **Activate Service** option.

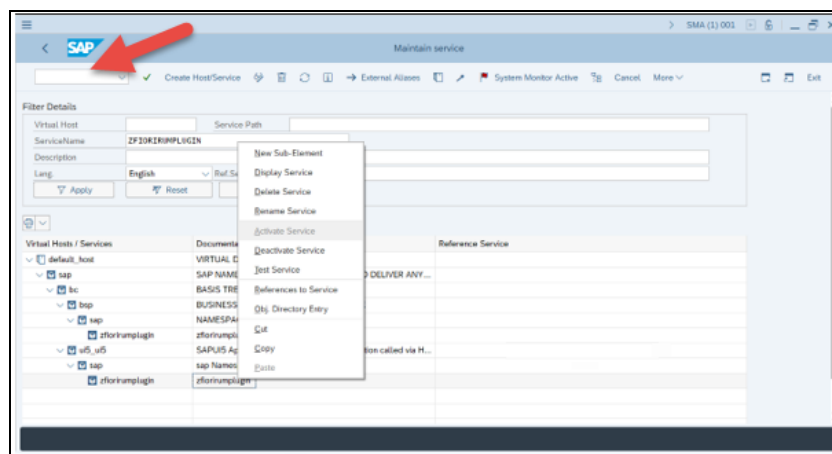


Figure 3.193: Activating the service

36. Next, let us proceed to create a catalog. For that, in the text area indicated by Figure 3.193, type the transaction code, `/n/UI2/FLPD_CUST`, and press the **Enter** key on your keyboard. This transaction will open in a web browser. To create a catalog, click on the encircled '+' icon indicated by Figure 3.194.

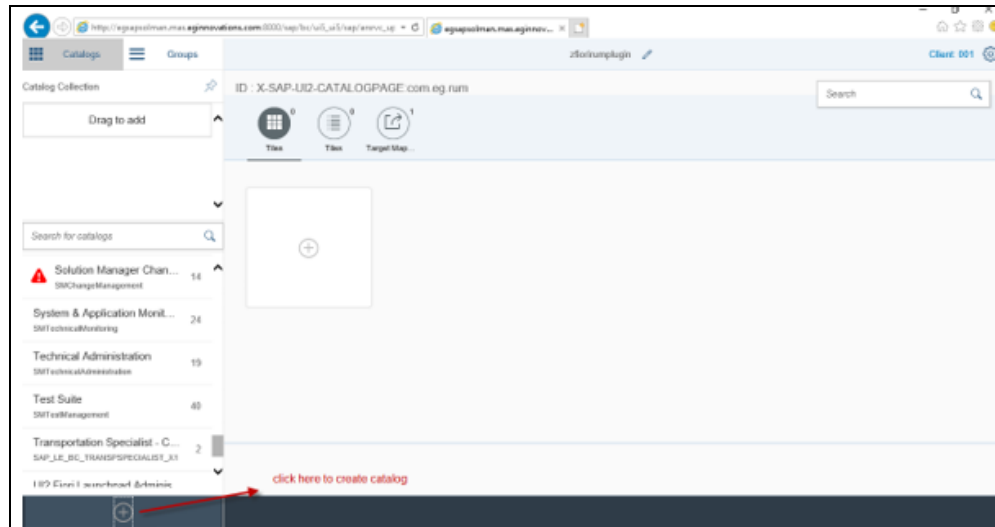
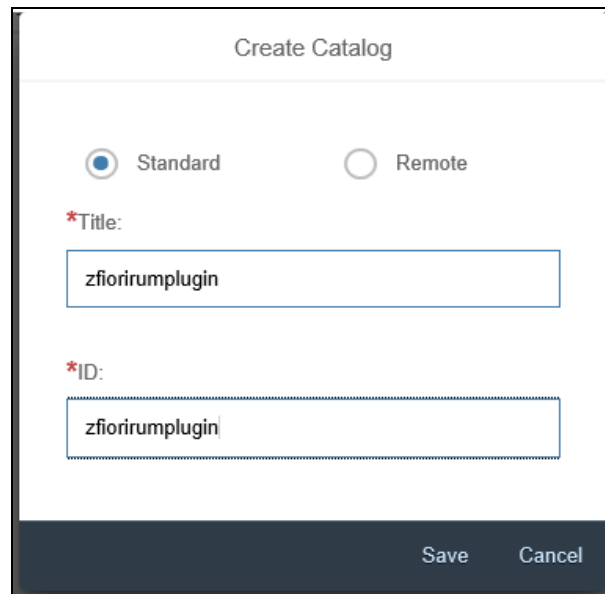


Figure 3.194: Clicking on the + icon to create a new catalog

37. When Figure 3.195 appears, specify the catalog **Title** (i.e., name of the new catalog), and assign a unique **ID** to the catalog. For easy identification, it is recommended that you provide the service name / application name you previously created as the catalog name. Then, click **Save** to save the changes.



The 'Create Catalog' dialog box is shown. It has two radio buttons: 'Standard' (selected) and 'Remote'. Below them are two text input fields. The first is labeled '*Title:' and contains the text 'zfiorumplugin'. The second is labeled '*ID:' and also contains the text 'zfiorumplugin'. At the bottom right are 'Save' and 'Cancel' buttons.

Figure 3.195: Creating a new catalog

38. Once the catalog is created, you will find the new catalog listed in the **Catalog Collection** panel of Figure 3.196. Double click on that catalog to open it. Then, click on the **Target Mapping** option indicated by Figure 3.196.

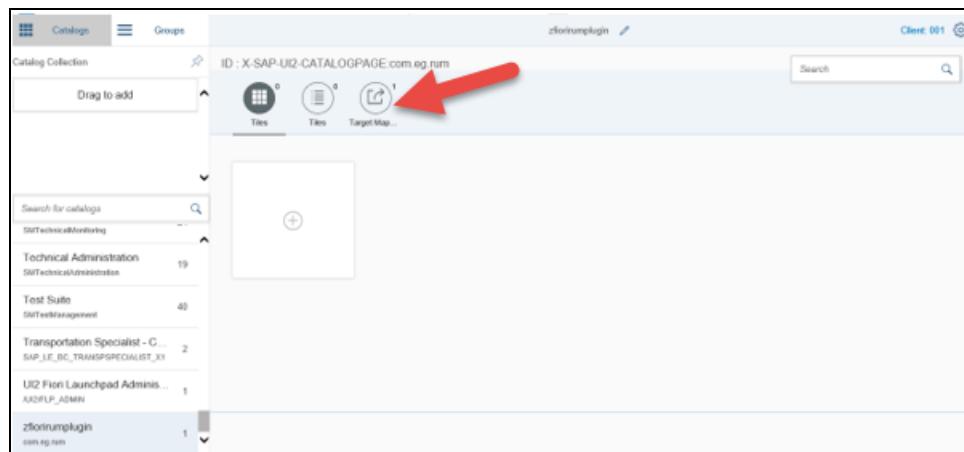


Figure 3.196: Clicking on the Target Mapping option

39. Then, click on the **Create Target Mapping** option indicated by Figure 3.197.

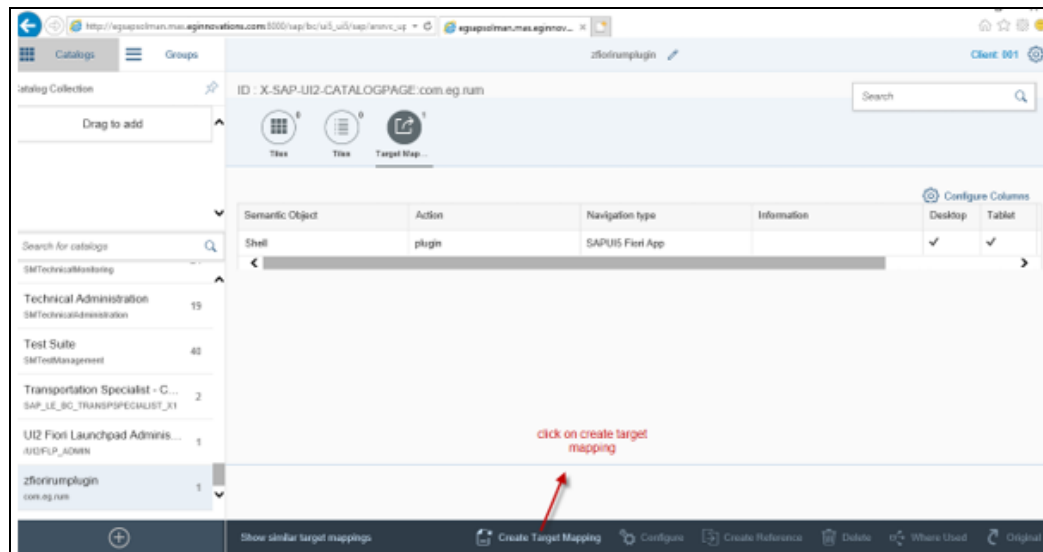


Figure 3.197: Clicking on the Create Target Mapping option

40. Configure the target mapping using Figure 3.198 that appears. Here, select *Shell* as the **Semantic Object** and *plugin* as **Action**. In **Title**, enter the application name of the eG RUM Plugin. Against **URL**, specify the path to the service that was automatically created after the eG RUM application was deployed (see step 34 for details). To know the path, look up the **Virtual Hosts/Services** tree-structure in Figure 3.193. Even if the eG RUM service is deployed in many locations, the **URL** should contain the path to the service in the *ui5_ui5* location only. As per Figure 3.193 therefore, the **URL** will be */sap/bc/ui5_ui5/sap/ zfiorumplugin*. Against **ID**, specify the **Plugin ID** that you provided at step 24 above (see Figure 3.182).

Configure 'zforinplugin' Instance ID: 005EJECFONL36E87XZ88GD8

Intent

Semantic Object:

Action:

Target

Application Type:

Title:

URL:

ID:

General

Information:

Device Types: ☒ Desktop ☒ Tablet ☒ Phone

Name	Mandatory	Value	Is Regular Expression	Default Value	Target N...
Enter a nam	☐		☐		

Save Cancel

Figure 3.198: Creating a new target mapping

41. Finally, click **Save** in Figure 3.198 to save the catalog. Next, proceed to assign the catalog to a user role. For that, in the text box indicated by Figure 3.193 above, specify *PFCG* and press the **Enter** key on the keyboard. When Figure 3.199 appears, provide a name to the new role in the **Role** text box, and click the **Single Role** button.

SAP Role Maintenance

Role:

Short Description:

☒ Single Role ☐ Comp. Role

Favorites	Description	Target Sys

Figure 3.199: Creating a new role

42. Figure 3.200 then appears. Click the **Menu** option indicated by Figure 3.200. Under **Hierarchy**, you will find **Role Menu**. Right-click on **Role Menu** and select **Assign Catalog** from the shortcut menu that appears. A list of catalogs will appear. Select the new catalog you created using steps 36 and 37 above. This will assign that catalog to the role.

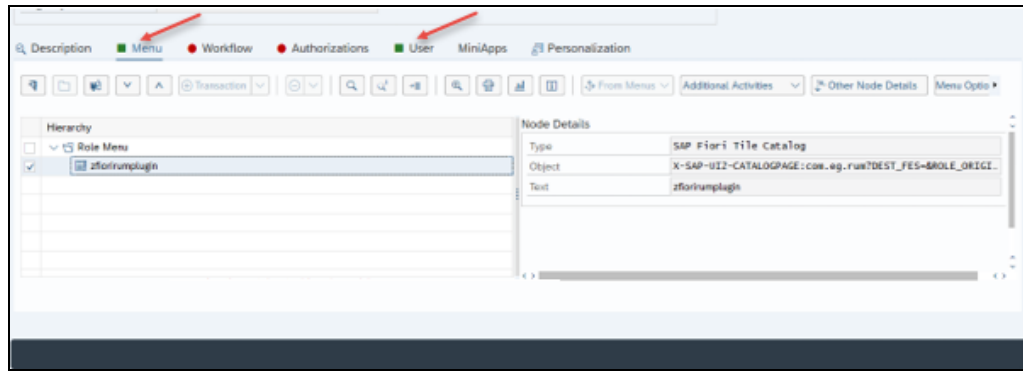


Figure 3.200: Assigning the newly created catalog to the new role

43. Next, click on the **User** option indicated by Figure 3.200 above to assign the new role to one/more Fiori users.
44. Finally, save all the changes.
45. Now, proceed to verify whether the SAP Fiori Apps has indeed been RUM-enabled or not. For that, first wait for the SAP Fiori Apps you added as a Real User Monitor component in eG Enterprise to start reporting metrics. Then, open the Fiori Launchpad. Click on the F12 key and refresh it to reload. If you find a GET request as indicated by Figure 3.201, you can conclude that the egrum.js has been successfully loaded. If you find a POST request bearing the RUM collector's IP and port number (as indicated by Figure 3.201), it means that the browser is sending beacons to the eG RUM collector.

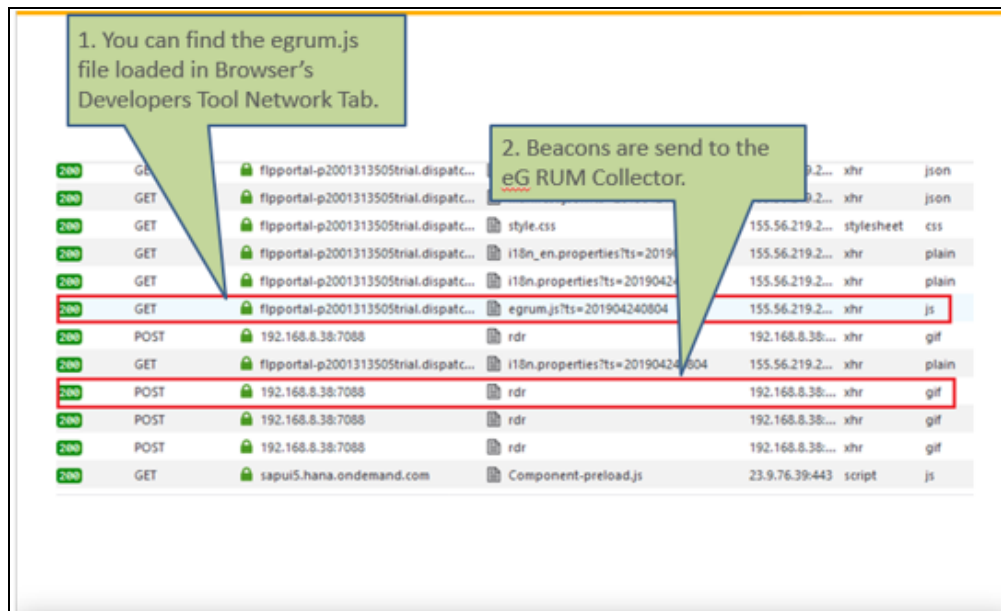


Figure 3.201: Verifying whether the eG RUM configuration for SAP Fiori Apps for On-Premise setup is working or not

3.4.13 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 3.202) 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 3.202 below.



```

1 <html>
2 <head>
3 <title>Blank page!!!</title>
4 <!-- RUM Header -->
5 <script charset='UTF-8' type='text/javascript'> window['egrum-start_time'] = new Date().getTime();
6 window['Site Name'] = 'EasyKart';
7 window['beacon-url'] = 'https://rum.eginnovations.com';</script>
8 <script src='https://rum.eginnovations.com/rumcollector/egrum.js' async></script>
9 <!-- RUM Header -->
10 <link rel="stylesheet" href="/final/config/styles/ec_style_green.css" />
11 <body leftmargin="0" topmargin="0" marginwidth="0" marginheight="0">
12 <table width="100%" border="0" cellspacing="0" cellpadding="0" align="center" height="100%" id="body_text">
13 <tr>
14 <td align="center" valign="middle">
15 <table width="72%" border="0" cellspacing="1" cellpadding="5" align="center">
16 <tr>
17 <td align="center" valign="top"><b>eG Configuration Help Pages!!!!!!!!!!</b></td>
18 </tr>
19 </table>
20 </td>
21 </tr>
22 </table>
23 </body>
24 </html>

```

Figure 3.202: 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 3.203).

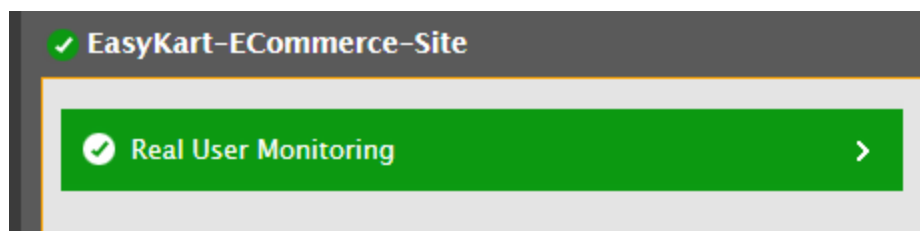


Figure 3.203: Layer model of the eG RUM component

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 3.203, will reveal these tests.

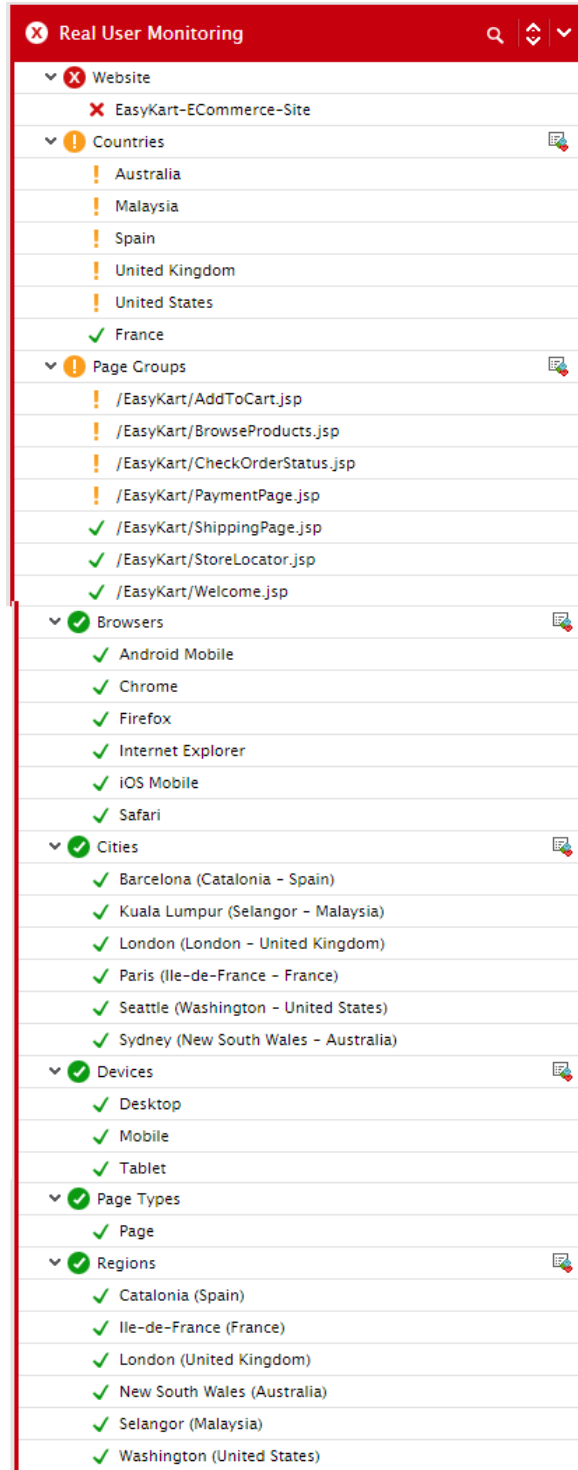


Figure 3.204: Tests mapped to the Real User Monitoring layer

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 3.205 below depicts how a web page loads.

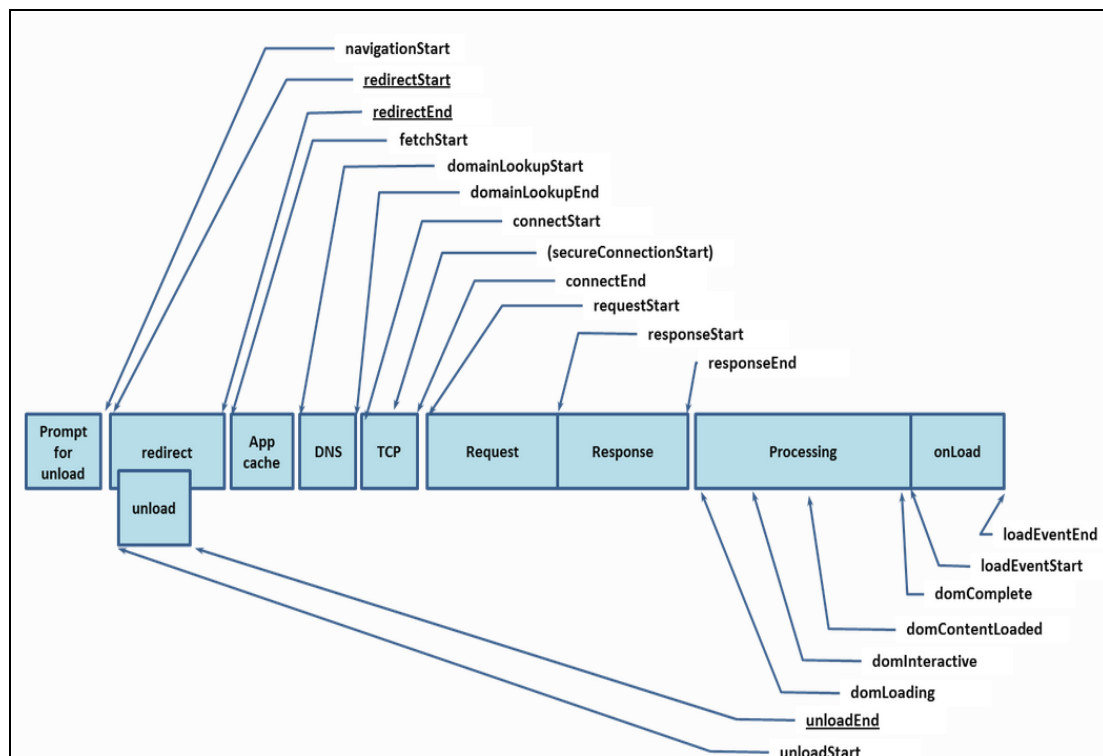


Figure 3.205: The page loading process

As is evident from Figure 3.205, 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

Configurable parameters for the test

Parameter	Description
Test Period	How often should the test be executed.
Proxy Host	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 <i>none</i> , indicating that the eG agent does not communicate with the collector via a proxy.
Proxy Port	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 <i>none</i> , indicating that the eG agent does not communicate with the collector via a proxy.
Proxy Username and Proxy Password	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 Proxy Username and Proxy Password . If no proxy server is used or if the proxy server used does not require authentication, then set the Proxy Username and Proxy Password to <i>none</i> .
Confirm Password	Confirm the Proxy Password by retyping it here. Note: If you Reconfigure the test later to change the values of the Proxy Username , Proxy Password , 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

Parameter	Description
	Yes. This will invoke the option depicted by the below figure.  Configuring the number of allowed page visits By default, the Maximum allowed page visits per day is set to <i>100000</i>. This implies that the test will consider only the first <i>100000</i> requests in a day for monitoring. All page visits beyond <i>100000</i> will by default be excluded from the test's monitoring purview. You can increase or decrease this limit, if you so need.
Consider JavaScript errors in Apdex calculations	The formula for computing the Apdex score is as follows: $[Satisfied\ requests + (Tolerating\ requests/2)]/Total\ samples$ By default, the count of <i>Satisfied</i> and <i>Tolerating requests</i> used for Apdex score computation includes the count of requests that experienced JavaScript errors. This is why, the Consider JavaScript errors in Apdex calculations flag is set to Yes by default. Since JavaScript errors also impact user experience, this default setting will result in an Apdex score that presents a near-real picture of the performance of a web site/web application. If you do not want the Apdex score to be tainted by error requests, then set this flag to <i>No</i>. In this case, requests with JavaScript errors will not be considered for Apdex calculations, even if such requests have a response time that is well-within (satisfied) or in violation (tolerating) of the Slow Transaction Cutoff.
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.eazycart.com/web/shopping/login.jsp, then http://www.eazycart.com will be the base URL or domain, <i>/web</i> will be the first URL segment, <i>/shopping</i> will be the second URL segment, and <i>/login.jsp</i> 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 <i>/web/shopping</i>. Here, <i>/web</i> corresponds to the specification 1 (URL segment 1) and <i>/shopping</i> corresponds to the specification 2 (URL segment 2) in the

Parameter	Description
	default value 1,2. This in turn means that, if the web site consists of pages such as http://www.eazycart.com/web/shopping/products.jsp, http://www.eazycart.com/web/shopping/products/travel/bags.jsp, http://www.eazycart.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 <i>/web/shopping</i>. This way, the test will create different page groups based on each of the second-level URL segments in the managed web site – eg., <i>/web/movies</i>, <i>/web/travel</i>, <i>/web/reservations</i>, <i>/partner/contacts</i> 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.eazycart.com/web/shopping/login.jsp, the test will report metrics for the descriptor <i>/shopping</i>. You can even set this parameter to 1,3. For a web site that contains URLs such as http://www.eazycart.com/web/shopping/products.jsp, http://www.eazycart.com/web/shopping/products/travel/bags.jsp, and http://www.eazycart.com/web/shopping/payment.jsp, this specification will result in the following descriptors: <i>/web/products</i>, <i>/web/products.jsp</i>, and <i>/web/payment.jsp</i>.
URL Patterns to be Ignored from Monitoring	By default, this test does not track requests to the following URL patterns: <i>*.js, *.css, *.jpeg, *.jpg, *.png</i>. 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 <i>.gif</i> and <i>.bmp</i> when monitoring, you need to alter the default specification as follows: <i>*.js, *.css, *.jpeg, *.jpg, *.png, *.gif, *.bmp</i> 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

Parameter	Description
	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 <i>none</i> 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: <i><Javascript error message>:<URL of the page/file where the message originated></i>. For instance, say that the <i>login.html</i> page in your web site runs a few Java scripts that throw <i>Object expected errors</i>, which you want the eG agent to ignore. In this case, your error specification can be as follows: <i>Object expected:http://www.eazycart.com/web/login.html</i>. Alternatively, you can provide only that text string with which the error message begins in your specification – eg., <i>Object:http://www.eazycart.com/web/login.html</i>. 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: <i>Object:login.html</i>. 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 <i>login.html</i> page points to a <i>validate.js</i> file, and you want to ignore the <i>Object expected</i> errors that this JS file throws, your error pattern specification will either be, <i>Object expected:validate.js</i>, or <i>Object:validate.js</i>. Multiple error message-URL combinations can also be provided as a comma-separated list. The format of your specification will be: <i><Javascript error message 1>:<Originating URL 1>,<Javascript error message 2>:<Originating URL 2>,...</i> For example, to ignore the <i>Object expected</i> and <i>Uncaught TypeError</i> errors in the <i>login.html</i> page, use the following specification:

Parameter	Description
	<i>Object:login.html,Uncaught:login.html</i> Likewise, to ignore the <i>Object expected</i> error in the <i>login.html</i> page and the <i>Uncaught TypeError</i> in the <i>validate.js</i> file, your specification will be: <i>Object:login.html,Uncaught:validate.js</i> If you want to ignore the <i>Uncaught TypeError</i> across all the pages of the web site, your specification will be as follows: <i>Uncaught TypeError:All</i> Note: When specifying the <i><Javascript error message></i> 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 <i>case-sensitive</i> pattern matching.
Include Page Type Categorization in Top-N DD	By default, the detailed diagnosis of this test reports the top-n transactions (in terms of <i>Avg page load time</i>) across page types. Accordingly, the Include Page Type Categorization in Top-N DD flag is set to No by default. If you want detailed diagnosis to report the top-n transactions per page type, then set this flag to Yes. In this case, if say the Maximum Healthy Transactions, Maximum Slow Transactions, and Maximum Error Transactions parameters are each set to 5, then the detailed diagnosis of this test will report the top-5 healthy, top-5 slow, and top-5 error transactions to Ajax pages, iFrames, Virtual pages, and base pages. As this may typically generate a large volume of data and increase the load on the eG database, you may want to set the 'Include Page Type Categorization in Top-N DD' to 'Yes', only if your eG database is sized sufficiently.
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, for each measurement period in the timeline chosen for detailed diagnosis. To identify the top 5, the eG agent sorts all healthy transactions in a measurement period 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

Parameter	Description
	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 <i>0</i>. To view all healthy transactions, set the value of this parameter to <i>all</i>. Before setting the value of this parameter to <i>all</i>, make sure that you have a well-sized eG database in place, as this setting will store details of every transaction that registers an <i>Avg page load time</i> 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 <i>5</i> indicating that the detailed diagnosis of this test will display metrics related to the top-5 slow transactions, in terms of the <i>Average page load time</i>, for each measurement period in the timeline chosen for detailed diagnosis. To identify the top 5, the eG agent sorts all slow transactions in a measurement period in the descending order of their <i>Average page load time</i>, 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 <i>0</i>. To view all slow transactions, set the value of this parameter to <i>all</i>. Before setting the value of this parameter to <i>all</i>, make sure that you have a well-sized eG database in place, as this setting will store details of every transaction that registers an <i>Avg page load time</i> 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 <i>5</i>, indicating that the detailed diagnosis of this test will display metrics related to the top-5 transactions (per measurement period) that encountered JavaScript errors, based on when those errors occurred. To identify the top 5, the eG agent sorts all error transactions in a measurement period 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

Parameter	Description
	of this parameter to <i>0</i>. To view all error transactions, set the value of this parameter to <i>all</i>. Before setting the value of this parameter to <i>all</i>, 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 <i>4000 millisecs</i> (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 <i>Maximum threshold</i> setting of the <i>Avg page load time</i> measure – i.e., both are set to <i>4000 millisecs</i> 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 <i>Avg page load time</i>. 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 Transaction Cutoff is changed to <i>6000 millisecs</i>, change the <i>Maximum Threshold</i> of the <i>Avg page load time</i> measure to <i>6000 millisecs</i> as well.
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, Virtual Page, 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 <i>3:1</i>. This indicates that, by default, detailed measures will be generated every third 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 <i>none</i> 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

Parameter	Description
	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 <i>Real User Monitor</i> 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

Measurement	Description	Measurement Unit	Interpretation
			current measurement period. In reality, these metrics pertain to the entire time period when the eG agent was unavailable. Because of this, the <i>Page views</i> 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 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 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 <i>Apdex score</i> in such circumstances to determine whether/not user experience has already been affected. Regardless, 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 <i>Average browser initial request time</i>, <i>Average network time</i>, <i>Average server time</i>, <i>Average content</i>

Measurement	Description	Measurement Unit	Interpretation
			<i>download time</i>, and <i>Average browser render time</i> measures of this test. If the <i>Average browser initial request time</i> is the highest, it can be attributed to a slowdown in redirection, AppCaching, or because the request spent too much time waiting on the browser. If the <i>Average network time</i> is the highest, it denotes that the network is the problem source. This in turn can be caused by TCP connection latencies, delays in domain look up, and slowdown in SSL handshake. If the <i>Average content download time</i> measure reports an abnormally high value, then it could be owing to slowness in downloading the HTML document or in building the DOM. On the other hand, if the <i>Average server time</i> 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.
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.
Unique user session	Indicates the number of distinct users who are	Number	HttpOnly is a flag that can be set on cookies to prevent their contents from

Measurement	Description	Measurement Unit	Interpretation
	currently accessing this web site/web application.		being accessed by JavaScript. This is often done for session cookies to hide the session identifier, as a security measure. If the HttpOnly flag is set for the session cookies of the monitored web site, then the eG RUM JavaScript will not be able to read the session identifier from the cookies. In such cases therefore, the eG agent will not be able to report the count of unique sessions.
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 <i>Slow page view percentage</i> with that of the <i>Javascript error page view percentage</i>. 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

Measurement	Description	Measurement Unit	Interpretation
			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 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 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

Measurement	Description	Measurement Unit	Interpretation
			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 <i>Page views</i> 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 <i>Tolerating page views</i> and the <i>Frustrated page views</i>, it is a clear indicator that web site performance is below-par. In such a case, use the detailed diagnosis of the <i>Tolerating page views</i> and <i>Frustrated page views</i> 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 Cutoff configuration of this test, but is less than 4 times the Slow Transaction Cutoff (i.e., $< 4 * \text{slow transaction cutoff}$), then such a page view is considered to be a <i>Tolerating page view</i>. Ideally, the value of this measure should be 0. A value higher than that of the <i>Satisfied page views</i> 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?

Measurement	Description	Measurement Unit	Interpretation
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 Cutoff configuration of this test(i.e., $> 4 * \text{Slow Transaction Cutoff}$), 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 <i>Satisfied page views</i> 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 application was 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	Number	To know which pages in the web site were accessed by tablet users and to evaluate

Measurement	Description	Measurement Unit	Interpretation
	accessed from tablets.		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.
Average browser initial request time	Indicates the interval between when a request was received by this web site, when it followed a redirect, to when it was processed in the AppCache.	ms	In Figure 3.205, the <i>Average browser initial request time</i> is the time spent between the navigationStart event to the domainLookupStart event. This also includes the time spent by the browser waiting for one event to end and the next to begin. In short, this measure is the sum of <i>Average redirect time</i>, <i>Average AppCache time</i>, and <i>Average browser wait time</i> measures. This means that if a request takes too long to follow a redirection, or if the AppCache takes too long to process the request, or if the request spends too much time on the browser for the previous request to complete, then the value of this measure will significantly increase. This in turn will impact user experience, and consequently, the <i>Apdex score</i>. This is why, if this measure reports an abnormal value, it is important that you compare the value of the <i>Average redirect time</i>, <i>Average AppCache time</i>, and <i>Average browser wait time</i> measures, to figure out where the request spent maximum time - in redirection? in the AppCache? or when waiting on the browser?
Average server time	Indicates the interval between the start of processing of a request on the browser to when the browser receives the	ms	In Figure 3.205, the <i>Average server time</i> is the time spent between the requestStart event and responseStart event. Ideally, a low value is desired for this

Measurement	Description	Measurement Unit	Interpretation
	response.		measure, as high values will certainly hurt the <i>Apdex score</i> 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 <i>Average server time</i> will increase, leaving users with an unsatisfactory experience with the web site/web application. Note: This test uses the Navigation Timing API to measure web site performance. The Navigation Timing API typically exposes several properties that offer information about the time at which different page load events happen (see Figure 3.205) - eg., the <i>requestStart</i> event, the <i>responseStart</i> event, etc. This test uses the time stamps provided by the Navigation Timing API to compute and report the duration of page load events, so you can accurately identify where page loading is bottlenecked. The Navigation Timing API on Internet Explorer (IE) v11 reports an incorrect time stamp for the <i>requestStart</i> event of the page loading process (see Figure 3.205). As a result, for page view requests

Measurement	Description	Measurement Unit	Interpretation
			initiated from IE 11 browsers alone, eG RUM will report incorrect values for this measure. This issue was noticed in IE 11 in April 2019. It is recommended that you track hot fixes/patches released by Microsoft post April 2019, study the release notes of such fixes/patches, and determine if this bug has been fixed in any. If so, then you are advised to apply that fix/patch on IE 11 to resolve the issue. Until then, we recommend that you use the following workaround to accurately measure the <i>Average server time</i> of a page view request. • Deploy eG BTM (Java/.NET, as the case may be) on the backend server hosting the target web site/web application. • Use eG BTM to trace the path of transactions to the target web site/web application and enable the (Java or .NET) Business Transactions test to capture metrics on transaction performance. • Next, use the detailed diagnostics reported by eG RUM to identify the page view requests coming from the IE 11 browser. Make a note of the values in the Request time, URL and Query Params columns of detailed diagnosis.

Measurement	Description	Measurement Unit	Interpretation
			- Then, search the detailed diagnostics of the (Java or .NET) Business Transactions test for transactions with the same URL, Request time, and Query Params as reported by eG RUM. - The response time that eG BTM reports for each of these transactions is the <i>Average server time</i> of those transactions. Note that this workaround applies only for those transaction URLs that are captured and reported as part of detailed diagnostics.
Average network 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 3.205, the time spent between <i>navigationStart</i> and <i>requestStart</i> makes up the <i>Average network time</i>. This includes the time to perform DNS lookups, the time to establish a TCP connection with the server, and the time to perform an SSL handshake. In other words, the value of this measure is nothing but the sum of the <i>Average DNS lookup time</i>, the <i>Average TCP connection time</i>, and the <i>Average SSL handshake time</i> 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 network time is high therefore, simply compare the values of the <i>Average DNS lookup time</i>, <i>Average TCP connection time</i>, and <i>Average SSL handshake time</i>

Measurement	Description	Measurement Unit	Interpretation
			measures to know to what this delay can be attributed – a delay in domain name resolution? a poor network connection to the server? or slowness in SSL negotiations between the browser and the server? Note: This test uses the Navigation Timing API to measure web site performance. The Navigation Timing API typically exposes several properties that offer information about the time at which different page load events happen (see Figure 3.205) - eg., the <i>requestStart</i> event, the <i>responseStart</i> event, etc. This test uses the time stamps provided by the Navigation Timing API to compute and report the duration of page load events, so you can accurately identify where page loading is bottlenecked. The Navigation Timing API on Internet Explorer (IE) v11 reports an incorrect time stamp for the <i>requestStart</i> event of the page loading process (see Figure 3.205). As a result, for page view requests initiated from IE 11 browsers alone, eG RUM will report incorrect values for this measure. This issue was noticed in IE 11 in April 2019. It is recommended that you track hot fixes/patches released by Microsoft post April 2019, study the release notes of such fixes/patches, and determine if this bug has been fixed in any. If so, then you are advised to apply that fix/patch on IE 11 to resolve the issue.
Average content	Indicates the time to	ms	This is the time spent between the

Measurement	Description	Measurement Unit	Interpretation
download time	make the complete HTML document (DOM) available for JavaScript to apply rendering logic.		responseStart event (shown in Figure 3.205) and the documentContentLoadedEventStart event (not shown in Figure 3.205). The documentContentLoadedEventStart is typically fired just before the domContentLoaded event, which is just after browser has finished downloading and parsing all the scripts that had defer set and no async attribute. In summary, the <i>Average content download time</i> measure is the sum of the values of the <i>Average DOM download time</i> and <i>Average DOM processing time</i> measures. This means that an abnormal increase in any of the above-mentioned time values will increase the value of this measure. If content downloading takes unusually long for a request, then you must compare the values of the <i>Average DOM download time</i> and <i>Average DOM processing time</i> measures to figure out what is causing the delay - is it because of the poor responsiveness of server, cache, or local resource? or is it because DOM processing took too long?
Average browser render time	Indicates the time taken to complete the download of remaining resources, including images, and to finish rendering the page.	ms	A high value of this measure indicates that page rendering is taking too long. This can be attributed to an in-optimal HTML document architecture, complex CSS' (eg., deeply nested rules, slow selectors, complicated effects such as round borders, gradients etc.), and large sized images.

Measurement	Description	Measurement Unit	Interpretation
			If the <i>Average page load time</i> measure reports an abnormally high value, then you may want to compare the value of this measure with that of the <i>Average browser initial request time</i> , <i>Average network time</i> , <i>Average server time</i> , <i>Average content download time</i> , and <i>Average browser rendering time</i> measures to nail the exact source of the bottleneck.
Average DOM download time	Indicates the time for the browser to download the complete HTML document content.	ms	The value of this measure is the time that elapsed between the responseStart and responseEnd events in Figure 3.205. 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. Consequently, content download will be delayed, thus adversely impacting user experience with the web site/web application. Ideally therefore, the value of this measure should be low.
Average browser wait time	Indicates the time spent by a request on the browser, waiting for another request to complete.	ms	This is the sum of the time between every two consecutive events, starting with the navigationStart event till the requestStart event in Figure 3.205. Typically, web browsers limit the number of active connections for each domain. Most modern browsers (eg., Chrome) support only six simultaneous

Measurement	Description	Measurement Unit	Interpretation
			requests/connections. In this case therefore, when the seventh request comes in, that request waits on the browser until the six requests sent previously are processed. The waiting time of the seventh request is the browser wait time. High browser wait time can prolong the browser's initial request time, thus adversely impacting the overall responsiveness of the web site/application. This is why, if the <i>Average browser initial request time</i> measure reports an abnormally high value, you will have to compare the values of the <i>Average browser wait time</i>, <i>Average redirection time</i>, and <i>Average AppCache time</i> measure to determine whether/not the initial request delay observed on the browser is because requests have been waiting on the browser for too long a time. Some of the means by which you can reduce browser waits are briefly discussed below: • Browsers such as Mozilla Firefox support upto 10 parallel requests. You may want to recommend such browsers for your web site/web application users, so that more requests are processed and fewer requests are queued on the browser, thus reducing browser wait time. • Web site/application developers can

Measurement	Description	Measurement Unit	Interpretation
			try domain sharding - i.e., split content across multiple domains. Typically, when a user connects to a web page, his or her browser scans the resulting HTML for resources to download. Normally these resources are supplied by a single domain - the domain providing the web page or a domain created specifically for resources. With domain sharding, the user's browser connects to two or more different domains to simultaneously download the resources needed to render the web page. This allows website/application to be delivered faster to users as they do not have to wait for the previous set of requests to end before beginning the next set.
Average redirection time	Indicates the time spent in redirection before fetching the page.	ms	In Figure 3.205, this is the elapsed time between the redirectStart and redirectEnd events. URL redirection, also known as URL forwarding, is a technique to give a page, a form, or a whole web site/application, more than one URL address. Usually, web site administrators use URL redirection to: • Redirect users to the mobile version of the site • Redirect users to secured pages • Redirect users to the latest version of

Measurement	Description	Measurement Unit	Interpretation
			the resource/content • Redirect users to pages specific to their geo location • Redirecting canonical URLs Though redirects are useful, they have to be kept at a minimum, as each redirect on a page adds latency to the overall page load time. This is because, when a user enters a domain into the browser and hits enter, the DNS resolution process is triggered and the domain is resolved to its corresponding IP address in a few milliseconds. If the landing page has another redirect, then the browser repeats the entire DNS resolution process once again to guide the user to the correct web page. The multiple redirect requests are taxing on the browser resources and slow down the page load. Web page load time is also affected by internal redirects; for example, if the page tries to load content from a URL that has been redirected to newer or updated content, then the browser must create additional requests to fetch the valid content. These redirects result in additional round trips between the browser and the web server which pushes the load time higher; the perceived performance is degraded every time the browser encounters a redundant redirect. Web site/application performance is also impacted if redirects are not implemented correctly. Some of the common redirect errors are:

Measurement	Description	Measurement Unit	Interpretation
			- Multiple redirects: The higher the number of redirects on a page, the higher is its page load time. - Invalid redirects: There are often instances where the web site administrator sets up bulk redirects without verifying the validity of the redirects. The site may also have old redirects that were never cleaned up. This can cause several issues on the site like broken links and 404s. - Redirect loop: When there are several redirects on the page that are linked to each other, it creates a chain of redirects which may loop back to the same URL that initiated the redirect. This results in a redirect loop error and the user will not be able to access the site. Therefore, if you find that the value of the <i>Average page load time</i> measure is abnormally high owing to an unusually high value for the Average redirection time measure, then make sure you follow the best practices outlined below while implementing redirects, so you can significantly reduce page load time and improve user experience: - Avoid redundant redirects: It's recommended to avoid redirects where possible and to use this method only

Measurement	Description	Measurement Unit	Interpretation
			when absolutely needed. This will cut down unnecessary overhead and improve the perceived performance of the page. • Chain redirects: When a URL is linked to another URL, this creates a chained redirect. Each URL added to the chain adds latency to the page. Chained redirects have a negative impact not only on page speed, but also SEO. • Clean-up redirects: You may have hundreds of redirects on your website and it could be one of the main factors affecting page speed. Old redirects may conflict with new URLs, backlinks can cause odd errors on the page. It is recommended to verify all the redirects you have set up and to remove the ones that are no longer needed. Retain the old links that have major referral traffic, while those that are rarely accessed can be removed. This exercise will help improve page speed significantly.
Average AppCache time	Indicates the time taken to check whether/not the requested URL is available in the AppCache.	ms	HTML5 provides an application caching mechanism that lets web-based applications run offline. Developers can use the Application Cache (AppCache) interface to specify resources that the browser should cache and make available to offline users. Applications that are cached load and work correctly even if users click the refresh button when they are offline. Using an application cache gives an application the following benefits:

Measurement	Description	Measurement Unit	Interpretation
			• Offline browsing: users can navigate a site even when they are offline. • Speed: cached resources are local, and therefore load faster. • Reduced server load: the browser only downloads resources that have changed from the server. To enable the application cache for an application, you must include the manifest attribute in the <html> element in your application's pages. The manifest attribute references a cache manifest file, which is a text file that lists resources (files) that the browser should cache for your application. The browser does not cache pages that do not contain the manifest attribute, unless such pages are explicitly listed in the manifest file itself. You do not need to list all the pages you want cached in the manifest file, the browser implicitly adds every page that the user visits and that has the manifest attribute set to the application cache. When the browser visits a document that includes the manifest attribute, if no application cache exists, the browser loads the document and then fetches all the entries listed in the manifest file, creating the first version of the application cache. Subsequent visits to that document cause the browser to load the document and other assets specified in the manifest file from the application cache. If the manifest file has changed, all the

Measurement	Description	Measurement Unit	Interpretation
			files listed in the manifest—as well as those already added to the cache—are fetched into a temporary cache. Once all the files have been successfully retrieved, they are moved into the real offline cache automatically. Since the document has already been loaded into the browser from the cache, the updated document will not be rendered until the document is reloaded (either manually or programmatically). A high value of this measure signifies that requests are spending too much time in the AppCache. This also introduces page loading latencies, which have an adverse effect on user-perceived performance of a web site/application. Common reasons for AppCaching issues and their practical solutions are detailed below: • If the media type is not set, then AppCache will not work. To avoid this, make sure that the manifest file is always served under the correct media type of text/cache-manifest. • If the manifest file is not served to the web browser from the same origin as the host page, the manifest file will fail to load. To avoid this, make sure that the manifest file is always served from the same origin as the host page. However, note that the manifest file can hold reference to resources to be cached from other domains.

Measurement	Description	Measurement Unit	Interpretation
			- The relative URLs that you mention in the manifest are relative to the manifest file and not to the document where you reference the manifest file. If you make this error when the manifest and the reference are not in the same path, the resources will fail to load and in turn the manifest file will not be loaded. This will stall appcaching. - Any change made to the manifest file will cause the entire set of files to be downloaded again. Moreover, if a manifest file is added to a HTML file, it forces all resources to be downloaded synchronously as soon as the manifest file is downloaded. As a result, resources that may not yet be required, such as JavaScript or an image below the fold, will be downloaded at the start of the page. This can increase page load time significantly. The solution to this is to load the Application Cache from a simple HTML file loaded in an iframe. This not only avoids caching dynamic HTML but also avoids the Application Cache being downloaded asynchronously after the page load has completed.
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 <i>Average network time</i> and <i>Average page load time</i> 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	ms	A bad network connection between the browser client and the server can delay

Measurement	Description	Measurement Unit	Interpretation
	establish a TCP connection with the server.		TCP connections to the server As a result, the <i>Average network time</i> too will increase, thus impacting page load time and overall user experience with the web site/web application.
Average SSL handshake time	Indicates the time taken to complete the SSL handshake.	ms	An SSL handshake happens when a browser makes a secure request for content, also known as an encrypted HTTPS connection. The user's browser and server negotiate encrypted keys and certificates to establish a secure connection between each other. Because this SSL negotiation requires exchanges between the browser and your server, it increases the time spent by the request on the network (i.e., it adds to the value of the Average network time measure). This in turn increases page load time. In fact, an SSL handshake, along with DNS lookup and TCP handshake add three round trips to the page load time. A quick fix to this is to use HTTP/2. HTTP/2 can use caching to reduce SSL setup to only one round trip.

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 used 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

Configurable parameters for the test

Parameter	Description
Test Period	How often should the test be executed.
Proxy Host	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 <i>none</i> , indicating that the eG agent does not communicate with the collector via a proxy.
Proxy Port	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 <i>none</i> , indicating that the eG agent does not communicate with the collector via a proxy.
Proxy Username and Proxy Password	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 Proxy Username and Proxy Password . If no proxy server is used or if the proxy server used does not require authentication, then set the ProxyUsername and ProxyPassword to <i>none</i> .

Parameter	Description
Confirm Password	Confirm the Proxy Password by retyping it here. Note: If you Reconfigure the test later to change the values of the Proxy Username, Proxy Password, 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. MAXIMUM ALLOWED PAGE VISITS PER DAY 100000 Configuring the number of allowed page visits By default, the Maximum allowed page visits per day is set to <i>100000</i>. This implies that the test will consider only the first <i>100000</i> requests in a day for monitoring. All page visits beyond <i>100000</i> will by default be excluded from the test's monitoring purview. You can increase or decrease this limit, if you so need.
Consider JavaScript errors in Apdex calculations	The formula for computing the Apdex score is as follows: $[Satisfied\ requests + (Tolerating\ requests/2)]/Total\ samples$ By default, the count of <i>Satisfied</i> and <i>Tolerating requests</i> used for Apdex score computation includes the count of requests that experienced JavaScript errors. This is why, the Consider JavaScript errors in Apdex calculations flag is set to Yes by default. Since JavaScript errors also impact user experience, this default setting will result in an Apdex score that presents a near-real picture of the performance of a web site/web application. If you do not want the Apdex score to be tainted by error requests, then set this flag to <i>No</i>. In this case, requests with JavaScript errors will not be considered for Apdex calculations, even if such requests have a response time that is well-within (satisfied) or in violation (tolerating) of the Slow Transaction Cutoff.
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

Parameter	Description
	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.eazycart.com/web/shopping/login.jsp, then http://www.eazycart.com will be the base URL or domain, <i>/web</i> will be the first URL segment, <i>/shopping</i> will be the second URL segment, and <i>/login.jsp</i> 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 <i>/web/shopping</i>. Here, <i>/web</i> corresponds to the specification 1 (URL segment 1) and <i>/shopping</i> 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.eazycart.com/web/shopping/products.jsp, http://www.eazycart.com/web/shopping/products/travel/bags.jsp, http://www.eazycart.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 <i>/web/shopping</i>. This way, the test will create different page groups based on each of the second-level URL segments in the managed web site – eg., <i>/web/movies</i>, <i>/web/travel</i>, <i>/web/reservations</i>, <i>/partner/contacts</i> 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.eazycart.com/web/shopping/login.jsp, the test will report metrics for the descriptor <i>/shopping</i>. You can even set this parameter to 1,3. For a web site that contains URLs such as http://www.eazycart.com/web/shopping/products.jsp, http://www.eazycart.com/web/shopping/products/travel/bags.jsp, and http://www.eazycart.com/web/shopping/payment.jsp, this specification will result in the following descriptors: <i>/web/products</i>, <i>/web/products.jsp</i>, and <i>/web/payment.jsp</i>.
URL Patterns to be Ignored from Monitoring	By default, this test does not track requests to the following URL patterns: <i>*.js, *.css, *.jpeg, *.jpg, *.png</i>. 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 <i>.gif</i> and <i>.bmp</i> when monitoring, you need to alter the default specification as follows: <i>*.js, *.css, *.jpeg, *.jpg, *.png, *.gif, *.bmp</i> 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

Parameter	Description
	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 <i>none</i> 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: <i><Javascript error message>:<URL of the page/file where the message originated></i>. For instance, say that the <i>login.html</i> page in your web site runs a few Java scripts that throw <i>Object expected errors</i>, which you want the eG agent to ignore. In this case, your error specification can be as follows: <i>Object expected:http://www.eazykart.com/web/login.html</i>. Alternatively, you can provide only that text string with which the error message begins in your specification – eg., <i>Object:http://www.eazykart.com/web/login.html</i>. 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: <i>Object:login.html</i>. 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

Parameter	Description
	is because, in this case, the file is where the error message originates. For instance, in the same example above, if the <i>login.html</i> page points to a <i>validate.js</i> file, and you want to ignore the <i>Object expected</i> errors that this JS file throws, your error pattern specification will either be, <i>Object expected:validate.js</i>, or <i>Object:validate.js</i>. Multiple error message-URL combinations can also be provided as a comma-separated list. The format of your specification will be: <pre><Javascript error message 1>:<Originating URL 1>,<Javascript error message 2>:<Originating URL 2>,. . .</pre> For example, to ignore the <i>Object expected</i> and <i>Uncaught TypeError</i> errors in the <i>login.html</i> page, use the following specification: <pre>Object:login.html,Uncaught:login.html</pre> Likewise, to ignore the <i>Object expected</i> error in the <i>login.html</i> page and the <i>Uncaught TypeError</i> in the <i>validate.js</i> file, your specification will be: <pre>Object:login.html,Uncaught:validate.js</pre> If you want to ignore the <i>Uncaught TypeError</i> across all the pages of the web site, your specification will be as follows: <pre>Uncaught TypeError:All</pre> Note: When specifying the <i><Javascript error message></i> 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 <i>case-sensitive</i> pattern matching.
Include Page Type Categorization in Top-N DD	By default, the detailed diagnosis of this test reports the top-n transactions (in terms of <i>Avg page load time</i>) across page types. Accordingly, the Include Page Type Categorization in Top-N DD flag is set to No by default. If you want detailed diagnosis to report the top-n transactions per page type, then set this flag to Yes. In this case, if say the Maximum Healthy Transactions, Maximum Slow Transactions, and Maximum Error Transactions parameters are each set to 5, then the detailed diagnosis of this test will report the top-5 healthy, top-5 slow, and top-5 error transactions to Ajax pages, iFrames, Virtual pages, and base pages. As this may typically generate a large volume of data and increase the load on the eG

Parameter	Description
	database, you may want to set the 'Include Page Type Categorization in Top-N DD' to 'Yes', only if your eG database is sized sufficiently.
Maximum Healthy Transactions in DD	By default, this parameter is set to <i>5</i> 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 <i>Average page load time</i>, for each measurement period in the timeline chosen for detailed diagnosis. To identify the top 5, the eG agent sorts all healthy transactions in a measurement period in the ascending order of their <i>Average page load time</i>, and picks the first <i>5</i> 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 <i>0</i>. To view all healthy transactions, set the value of this parameter to <i>all</i>. Before setting the value of this parameter to <i>all</i>, make sure that you have a well-sized eG database in place, as this setting will store details of every transaction that registers an <i>Avg page load time</i> 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 <i>5</i> indicating that the detailed diagnosis of this test will display metrics related to the top-5 slow transactions, in terms of the <i>Average page load time</i>, for each measurement period in the timeline chosen for detailed diagnosis. To identify the top 5, the eG agent sorts all slow transactions in a measurement period in the descending order of their <i>Average page load time</i>, and picks the first <i>5</i> 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 <i>0</i>. To view all slow transactions, set the value of this parameter to <i>all</i>. Before setting the value of this parameter to <i>all</i>, make sure that you have a well-sized eG database in place, as this setting will store details of every transaction that registers an <i>Avg page load time</i> 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	By default, this parameter is set to <i>5</i> , indicating that the detailed diagnosis of this test

Parameter	Description
Transactions in DD	will display metrics related to the top-5 transactions (per measurement period) that encountered JavaScript errors, based on when those errors occurred. To identify the top 5, the eG agent sorts all error transactions in a measurement period 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 <i>0</i>. To view all error transactions, set the value of this parameter to <i>all</i>. Before setting the value of this parameter to <i>all</i>, 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 <i>4000 millisecs</i> (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 <i>Maximum threshold</i> setting of the <i>Avg page load time</i> measure – i.e., both are set to <i>4000 millisecs</i> 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 <i>Avg page load time</i>. 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 Transaction Cutoff is changed to <i>6000 millisecs</i>, change the <i>Maximum Threshold</i> of the <i>Avg page load time</i> measure to <i>6000 millisecs</i> as well.
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, Virtual Page, and/or iFrame views as well in the dashboard, by selecting the relevant options from this list box.
Send Zero Values when there is	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 <i>0</i> - then

Parameter	Description
No Traffic	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 is no traffic to the monitored web site/web application.
DD Frequency	Refers to the frequency with which detailed diagnosis measures are to be generated for this test. The default is 3:1. This indicates that, by default, detailed measures will be generated every third 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 <i>none</i> 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 using this browser.	Number	This is a good measure of the traffic from a specific browser. 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.

Measurement	Description	Measurement Unit	Interpretation
			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 <i>Real User Monitor</i> 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 <i>Page views</i> 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 on 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

Measurement	Description	Measurement Unit	Interpretation
			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 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 using this browser has been less than satisfactory.
Average page load time	Indicates the average time that pages accessed using this browser 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 browser, you will be able to tell if the page load time is significantly higher

Measurement	Description	Measurement Unit	Interpretation
			for any one browser – this could be the browser that is causing the slowness. You may want to compare the values of the of the <i>Average browser initial request time</i>, <i>Average network time</i>, <i>Average server time</i>, <i>Average content download time</i>, and <i>Average browser render time</i> measures for that browser to know what is exactly causing users using that browser to experience slowness. If the <i>Average browser initial request time</i> is the highest, it can be attributed to a slowdown in redirection, AppCaching, or because the request spent too much time waiting on the browser. If the <i>Average network time</i> is the highest, it denotes that the network is the problem source. This in turn can be caused by TCP connection latencies, delays in domain look up, and slowdown in SSL handshake. If the <i>Average content download time</i> measure reports an abnormally high value, then it could be owing to slowness in downloading the HTML document or in building the DOM. On the other hand, if the <i>Average server time</i> 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. To know which pages accessed using this browser are slow, use the detailed diagnosis of this measure.
Unique user session	Indicates the number of distinct users who are currently accessing the	Number	

Measurement	Description	Measurement Unit	Interpretation
	web site/web application using this browser.		
Page views per minute	Indicates the number of times the pages were viewed per minute using this browser.	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 on this browser.	Percent	The value of this measure indicates the percentage of page views in which users using this browser 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 on 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 view requests are either slow or have encountered Javascript errors on this browser. Under such circumstances, to know what exactly is affecting the user experience on the browser, compare the value of the <i>Slow page view percentage</i> with that of the <i>Javascript error page view percentage</i> 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 <i>Slow page views</i> measure to know which pages accessed by users using this browser are slow and where these pages are losing time. If JavaScript errors are the problem, use the detailed diagnosis of the <i>JavaScript error view percentage</i> measure to know

Measurement	Description	Measurement Unit	Interpretation
			what errors occurred in which pages accessed using this browser.
Slow page view percentage	Indicates the percentage of pages that were slow in loading when accessed using 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 <i>Slow page views</i> 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 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 experiencing JavaScript errors.
Slow page views (Tolerating & Frustrated)	Indicates the number of slow page views 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 frequently experienced slowness when accessing pages using this browser. To know which pages were slow when accessed using this browser and why, use the detailed diagnosis of this measure.
JavaScript error page views	Indicates the number of times JavaScript errors occurred when accessing pages 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 pages 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!

Measurement	Description	Measurement Unit	Interpretation
Satisfied page views	Indicates the number of times pages were viewed on 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 <i>Page views</i> 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 <i>Tolerating page views</i> and the <i>Frustrated page views</i>, it is a clear indicator that the experience of users on this browser has been below-par. In such a case, use the detailed diagnosis of the <i>Tolerating page views</i> and <i>Frustrated page views</i> measures to know which pages are slow and why.
Tolerating page views	Indicates the number of tolerating page views experienced by users on this browser.	Number	If the <i>Average page load time</i> of a page exceeds the Slow Transaction Cutoff configuration of this test, but is less than 4 times the Slow Transaction Cutoff (i.e., $< 4 * \text{Slow Transaction Cutoff}$), then such a page view is considered to be a <i>Tolerating page view</i>. Ideally, the value of this measure should be 0. A value higher than that of the <i>Satisfied page views</i> measure is a cause for concern, as it implies that the overall experience of users on this browser is than satisfactory. To know which pages are contributing to this sub-par

Measurement	Description	Measurement Unit	Interpretation
			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 on this browser.	Number	If the <i>Average page load time</i> of a page is over 4 times the Slow Transaction Cutoff configuration of this test (i.e., $> 4 * \text{Slow Transaction Cutoff}$), 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 <i>Satisfied page views</i> measure is a cause for concern, as it implies that the experience of users has been less than satisfactory on this browser. 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 browser initial request time	Indicates the interval between when a request was received from users, when it followed a redirect, to when it was processed in the AppCache, on this browser.	ms	In Figure 3.205, the <i>Average browser initial request time</i> is the time spent between the navigationStart event to the domainLookupStart event. This also includes the time spent by the browser waiting for one event to end and the next to begin. In short, this measure is the sum of <i>Average redirect time</i> , <i>Average AppCache time</i> , and <i>Average browser wait time</i> measures. This means that if a request takes too long to follow a redirection, or if the AppCache takes too long to process the request, or if the

Measurement	Description	Measurement Unit	Interpretation
			request spends too much time on the browser for the previous request to complete, then the value of this measure will significantly increase. This in turn will impact user experience, and consequently, the <i>Apdex score</i>. This is why, if this measure reports an abnormal value for a browser, it is important that you compare the value of the <i>Average redirect time</i>, <i>Average AppCache time</i>, and <i>Average browser wait time</i> measures for the same browser, to figure out where the request spent maximum time - in redirection? in the AppCache? or when waiting on the browser?
Average server time	Indicates the interval between the start of processing of a request to when response is received on this browser.	ms	In Figure 3.205, the <i>Average server time</i> is the time spent between the <i>requestStart</i> event and <i>responseStart</i> event. Ideally, a low value is desired for this measure, as high values will certainly hurt the <i>Apdex score</i> 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 <i>Average server time</i> will increase, leaving users with an unsatisfactory

Measurement	Description	Measurement Unit	Interpretation
			experience with the web site/web application. Note: This test uses the Navigation Timing API to measure web site performance. The Navigation Timing API typically exposes several properties that offer information about the time at which different page load events happen (see Figure 3.205) - eg., the <code>requestStart</code> event, the <code>responseStart</code> event, etc. This test uses the time stamps provided by the Navigation Timing API to compute and report the duration of page load events, so you can accurately identify where page loading is bottlenecked. The Navigation Timing API on Internet Explorer (IE) v11 reports an incorrect time stamp for the <i>requestStart</i> event of the page loading process (see Figure 3.205). As a result, for page view requests initiated from IE 11 browsers alone, eG RUM will report incorrect values for this measure. This issue was noticed in IE 11 in April 2019. It is recommended that you track hot fixes/patches released by Microsoft post April 2019, study the release notes of such fixes/patches, and determine if this bug has been fixed in any. If so, then you are advised to apply that fix/patch on IE 11 to resolve the issue. Until then, we recommend that you use the following workaround to accurately measure the <i>Average server time</i> of a page view request. Deploy eG BTM (Java/.NET, as the

Measurement	Description	Measurement Unit	Interpretation
			case may be) on the backend server hosting the target web site/web application. - Use eG BTM to trace the path of transactions to the target web site/web application and enable the (Java or .NET) Business Transactions test to capture metrics on transaction performance. - Next, use the detailed diagnostics reported by eG RUM to identify the page view requests coming from the IE 11 browser. Make a note of the values in the Request time, URL and Query Params columns of detailed diagnosis. - Then, search the detailed diagnostics of the (Java or .NET) Business Transactions test for transactions with the same URL, Request time, and Query Params as reported by eG RUM. - The response time that eG BTM reports for each of these transactions is the <i>Average server time</i> of those transactions. Note that this workaround applies only for those transaction URLs that are captured and reported as part of detailed diagnostics.
Average network time	Indicates the elapsed time since a user initiates a request to the web site/application and	ms	In Figure 3.205, the time spent between <i>navigationStart</i> and <i>requestStart</i> makes up the <i>Average network time</i>. This includes the time to perform DNS

Measurement	Description	Measurement Unit	Interpretation
	the start of fetching the response document from it, on this browser.		lookups, the time to establish a TCP connection with the server, and the time to perform an SSL handshake. In other words, the value of this measure is nothing but the sum of the <i>Average DNS lookup time</i> , the <i>Average TCP connection time</i> , and the <i>Average SSL handshake time</i> 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 <i>Average DNS lookup time</i> , <i>Average TCP connection time</i>, and <i>Average SSL handshake time</i> measures to know to what this delay can be attributed – a delay in domain name resolution? Or a poor network connection to the server? or slowness in SSL negotiations between the browser and the server? Note: This test uses the Navigation Timing API to measure web site performance. The Navigation Timing API typically exposes several properties that offer information about the time at which different page load events happen (see Figure 3.205) - eg., the requestStart event, the responseStart event, etc. This test uses the time stamps provided by the Navigation Timing API to compute and report the duration of page load events, so you can accurately identify where page loading is bottlenecked. The Navigation Timing API on Internet

Measurement	Description	Measurement Unit	Interpretation
			Explorer (IE) v11 reports an incorrect time stamp for the <i>requestStart</i> event of the page loading process (see 3.6.2). As a result, for page view requests initiated from IE 11 browsers alone, eG RUM will report incorrect values for this measure. This issue was noticed in IE 11 in April 2019. It is recommended that you track hot fixes/patches released by Microsoft post April 2019, study the release notes of such fixes/patches, and determine if this bug has been fixed in any. If so, then you are advised to apply that fix/patch on IE 11 to resolve the issue.
Average content download time	Indicates the time to make the complete HTML document (DOM) available for JavaScript to apply rendering logic on the pages accessed using this browser.	ms	This is the time spent between the <i>responseStart</i> event (shown in 3.6.2) and the <i>documentContentLoadedEventStart</i> event (not shown in 3.6.2). The <i>documentContentLoadedEventStart</i> is typically fired just before the <i>domContentLoaded</i> event, which is just after browser has finished downloading and parsing all the scripts that had <i>defer</i> set and no <i>async</i> attribute. In summary, the <i>Average content download time</i> measure is the sum of the values of the <i>Average DOM download time</i> and <i>Average DOM processing time</i> measures. This means that an abnormal increase in any of the above-mentioned time values will increase the value of this measure. If content downloading takes unusually long for a request, then you must compare the values of the <i>Average DOM download time</i> and <i>Average DOM processing time</i>

Measurement	Description	Measurement Unit	Interpretation
			measures to figure out what is causing the delay - is it because of the poor responsiveness of server, cache, or local resource? or is it because DOM processing took too long?
Average browser render time	Indicates the time taken to complete the download of remaining resources, including images, and to finish rendering the pages accessed using this browser.	ms	A high value of this measure indicates that page rendering is taking too long. This can be attributed to an in-optimal HTML document architecture, complex CSS' (eg., deeply nested rules, slow selectors, complicated effects such as round borders, gradients etc.), and large sized images. If the <i>Average page load time</i> measure reports an abnormally high value, then you may want to compare the value of this measure with that of the <i>Average browser initial request time</i>, <i>Average network time</i>, <i>Average server time</i>, and <i>Average content download time</i> measures to nail the exact source of the bottleneck.
Average DOM download time	Indicates the time taken to download the complete HTML document for page view requests on this browser.	ms	The value of this measure is the time that elapsed between the responseStart and responseEnd events in 3.6.2. 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	Indicates the time taken	ms	An unusually high value for this measure

Measurement	Description	Measurement Unit	Interpretation
processing time	by the browser to build the Document Object Model (DOM) for the pages accessed using this browser and make it available for JavaScript to apply rendering logic.		is a clear indicator that DOM building is taking longer than normal. Consequently, content download will be delayed, thus adversely impacting user experience with the web site/web application. Ideally therefore, the value of this measure should be low.
Average browser wait time	Indicates the time that requests spent on this browser, waiting for another request to complete.	ms	This is the sum of the time between every two consecutive events, starting with the navigationStart event till the requestStart event in 3.6.2. Typically, web browsers limit the number of active connections for each domain. Most modern browsers (eg., Chrome) support only six simultaneous requests/connections. In this case therefore, when the seventh request comes in, that request waits on the browser until the six requests sent previously are processed. The waiting time of the seventh request is the browser wait time. High browser wait time can prolong the browser's initial request time, thus adversely impacting the overall responsiveness of the web site/application. This is why, if the <i>Average browser initial request time</i> measure reports an abnormally high value, you will have to compare the values of the <i>Average browser wait time</i>, <i>Average redirection time</i>, and <i>Average AppCache time</i> measure to determine whether/not the initial request delay observed on the browser is because requests have been waiting on the

Measurement	Description	Measurement Unit	Interpretation
			browser for too long a time. Some of the means by which you can reduce browser waits are briefly discussed below: • Browsers such as Mozilla Firefox support upto 10 parallel requests. You may want to recommend such browsers for your web site/web application users, so that more requests are processed and fewer requests are queued on the browser, thus reducing browser wait time. • Web site/application developers can try domain sharding - i.e., split content across multiple domains. Typically, when a user connects to a web page, his or her browser scans the resulting HTML for resources to download. Normally these resources are supplied by a single domain - the domain providing the web page or a domain created specifically for resources. With domain sharding, the user's browser connects to two or more different domains to simultaneously download the resources needed to render the web page. This allows website/application to be delivered faster to users as they do not have to wait for the previous set of requests to end before beginning the next set.

Measurement	Description	Measurement Unit	Interpretation
Average redirection time	Indicates the time that requests received from this browser spent in redirection before fetching the pages.	ms	In 3.6.2, this is the elapsed time between the redirectStart and redirectEnd events. URL redirection, also known as URL forwarding, is a technique to give a page, a form, or a whole web site/application, more than one URL address. Usually, web site administrators use URL redirection to: • Redirect users to the mobile version of the site • Redirect users to secured pages • Redirect users to the latest version of the resource/content • Redirect users to pages specific to their geo location • Redirecting canonical URLs Though redirects are useful, they have to be kept at a minimum, as each redirect on a page adds latency to the overall page load time. This is because, when a user enters a domain into the browser and hits enter, the DNS resolution process is triggered and the domain is resolved to its corresponding IP address in a few milliseconds. If the landing page has another redirect, then the browser repeats the entire DNS resolution process once again to guide the user to the correct web page. The multiple redirect requests are taxing on the browser resources and slow down the page load. Web page load time is also affected by internal redirects; for example, if the page

Measurement	Description	Measurement Unit	Interpretation
			tries to load content from a URL that has been redirected to newer or updated content, then the browser must create additional requests to fetch the valid content. These redirects result in additional round trips between the browser and the web server which pushes the load time higher; the perceived performance is degraded every time the browser encounters a redundant redirect. Web site/application performance is also impacted if redirects are not implemented correctly. Some of the common redirect errors are: • Multiple redirects: The higher the number of redirects on a page, the higher is its page load time. • Invalid redirects: There are often instances where the web site administrator sets up bulk redirects without verifying the validity of the redirects. The site may also have old redirects that were never cleaned up. This can cause several issues on the site like broken links and 404s. • Redirect loop: When there are several redirects on the page that are linked to each other, it creates a chain of redirects which may loop back to the same URL that initiated the redirect. This results in a redirect loop error and the user will not be able to access the site.

Measurement	Description	Measurement Unit	Interpretation
			Therefore, if you find that the value of the <i>Average page load time</i> measure is abnormally high owing to an unusually high value for the <i>Average redirection time</i> measure, then make sure you follow the best practices outlined below while implementing redirects, so you can significantly reduce page load time and improve user experience: • Avoid redundant redirects: It's recommended to avoid redirects where possible and to use this method only when absolutely needed. This will cut down unnecessary overhead and improve the perceived performance of the page. • Chain redirects: When a URL is linked to another URL, this creates a chained redirect. Each URL added to the chain adds latency to the page. Chained redirects have a negative impact not only on page speed, but also SEO. • Clean-up redirects: You may have hundreds of redirects on your website and it could be one of the main factors affecting page speed. Old redirects may conflict with new URLs, backlinks can cause odd errors on the page. It is recommended to verify all the redirects you have set up and to remove the ones that are no longer needed. Retain the old links that have major referral traffic, while those that are rarely accessed can be removed. This exercise will help improve page speed

Measurement	Description	Measurement Unit	Interpretation
			significantly.
Average AppCache time	Indicates the time taken to check whether/not the requests received from this browser can be serviced by the AppCache.	ms	HTML5 provides an application caching mechanism that lets web-based applications run offline. Developers can use the Application Cache (AppCache) interface to specify resources that the browser should cache and make available to offline users. Applications that are cached load and work correctly even if users click the refresh button when they are offline. Using an application cache gives an application the following benefits: • Offline browsing: users can navigate a site even when they are offline. • Speed: cached resources are local, and therefore load faster. • Reduced server load: the browser only downloads resources that have changed from the server. To enable the application cache for an application, you must include the manifest attribute in the <html> element in your application's pages. The manifest attribute references a cache manifest file, which is a text file that lists resources (files) that the browser should cache for your

Measurement	Description	Measurement Unit	Interpretation
			application. The browser does not cache pages that do not contain the manifest attribute, unless such pages are explicitly listed in the manifest file itself. You do not need to list all the pages you want cached in the manifest file, the browser implicitly adds every page that the user visits and that has the manifest attribute set to the application cache. When the browser visits a document that includes the manifest attribute, if no application cache exists, the browser loads the document and then fetches all the entries listed in the manifest file, creating the first version of the application cache. Subsequent visits to that document cause the browser to load the document and other assets specified in the manifest file from the application cache. If the manifest file has changed, all the files listed in the manifest—as well as those already added to the cache—are fetched into a temporary cache. Once all the files have been successfully retrieved, they are moved into the real offline cache automatically. Since the document has already been loaded into the browser from the cache, the updated document will not be rendered until the document is reloaded (either manually or programmatically). A high value of this measure signifies that requests are spending too much time in the AppCache. This also introduces page loading latencies, which have an adverse effect on user-perceived performance of a

Measurement	Description	Measurement Unit	Interpretation
			web site/application. Common reasons for AppCaching issues and their practical solutions are detailed below: • If the media type is not set, then AppCache will not work. To avoid this, make sure that the manifest file is always served under the correct media type of text/cache-manifest. • If the manifest file is not served to the web browser from the same origin as the host page, the manifest file will fail to load. To avoid this, make sure that the manifest file is always served from the same origin as the host page. However, note that the manifest file can hold reference to resources to be cached from other domains. • The relative URLs that you mention in the manifest are relative to the manifest file and not to the document where you reference the manifest file. If you make this error when the manifest and the reference are not in the same path, the resources will fail to load and in turn the manifest file will not be loaded. This will stall appcaching. • Any change made to the manifest file will cause the entire set of files to be downloaded again. Moreover, if a manifest file is added to a HTML file, it forces all resources to be downloaded

Measurement	Description	Measurement Unit	Interpretation
			synchronously as soon as the manifest file is downloaded. As a result, resources that may not yet be required, such as JavaScript or an image below the fold, will be downloaded at the start of the page. This can increase page load time significantly. The solution to this is to load the Application Cache from a simple HTML file loaded in an iframe. This not only avoids caching dynamic HTML but also avoids the Application Cache being downloaded asynchronously after the page load has completed.
Average DNS lookup time	Indicates the time taken by this browser to perform the domain lookup for connecting to the pages requested.	ms	A high value for this measure will not only affect DNS lookup, but will also impact the <i>Average network time</i> and <i>Average page load time</i> 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 for accessing pages.	ms	A bad network connection between the browser client and the server can delay TCP connections to the server. As a result, the <i>Average network time</i> too will increase, thus impacting page load time and overall user experience with the web site/web application.
Average SSL handshake time	Indicates the time taken by requests to complete the SSL handshake on this browser.	ms	An SSL handshake happens when a browser makes a secure request for content, also known as an encrypted HTTPS connection. The user's browser and server negotiate encrypted keys and certificates to establish a secure connection between each other. Because this SSL negotiation requires exchanges between the browser and your server, it increases the time spent by the request on the network (i.e., it adds to the value of the

Measurement	Description	Measurement Unit	Interpretation
			Average network time measure). This in turn increases page load time. In fact, an SSL handshake, along with DNS lookup and TCP handshake add three round trips to the page load time. A quick fix to this is to use HTTP/2. HTTP/2 can use caching to reduce SSL setup to only one round trip.

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 device used for accessing the web site/web application being monitored.

Configurable parameters for the test

Parameter	Description
Test Period	How often should the test be executed.
Proxy Host	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 <i>none</i> , indicating that the eG agent does not communicate with the collector via a proxy.
Proxy Port	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 <i>none</i> , indicating that the eG agent does not communicate with the collector via a proxy.
Proxy Username and Proxy Password	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 Proxy Username and Proxy Password . If no proxy server is used or if the proxy server used does not require authentication, then set the ProxyUsername and ProxyPassword to <i>none</i> .
Confirm Password	Confirm the Proxy Password by retyping it here.
	Note: If you Reconfigure the test later to change the values of the Proxy Username, Proxy Password, 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.

MAXIMUM ALLOWED PAGE VISITS PER DAY

100000

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.

Parameter	Description
	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.
Consider JavaScript errors in Apdex calculations	The formula for computing the Apdex score is as follows: $[Satisfied\ requests + (Tolerating\ requests/2)]/Total\ samples$ By default, the count of <i>Satisfied</i> and <i>Tolerating requests</i> used for Apdex score computation includes the count of requests that experienced JavaScript errors. This is why, the Consider JavaScript errors in Apdex calculations flag is set to Yes by default. Since JavaScript errors also impact user experience, this default setting will result in an Apdex score that presents a near-real picture of the performance of a web site/web application. If you do not want the Apdex score to be tainted by error requests, then set this flag to <i>No</i>. In this case, requests with JavaScript errors will not be considered for Apdex calculations, even if such requests have a response time that is well-within (satisfied) or in violation (tolerating) of the Slow Transaction Cutoff.
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.eazycart.com/web/shopping/login.jsp, then http://www.eazycart.com will be the base URL or domain, <i>/web</i> will be the first URL segment, <i>/shopping</i> will be the second URL segment, and <i>/login.jsp</i> 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 <i>/web/shopping</i>. Here, <i>/web</i> corresponds to the specification 1 (URL segment 1) and <i>/shopping</i> 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.eazycart.com/web/shopping/products.jsp, http://www.eazycart.com/web/shopping/products/travel/bags.jsp, http://www.eazycart.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 <i>/web/shopping</i>. This way, the test will create different page groups based on each of the second-level URL segments in the managed web site – eg., <i>/web/movies</i>, <i>/web/travel</i>, <i>/web/reservations</i>,

Parameter	Description
	<i>/partner/contacts</i> 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.eazycart.com/web/shopping/login.jsp, the test will report metrics for the descriptor <i>/shopping</i>. You can even set this parameter to 1,3. For a web site that contains URLs such as http://www.eazycart.com/web/shopping/products.jsp, http://www.eazycart.com/web/shopping/products/travel/bags.jsp, and http://www.eazycart.com/web/shopping/payment.jsp, this specification will result in the following descriptors: <i>/web/products</i>, <i>/web/products.jsp</i>, and <i>/web/payment.jsp</i>.
URL Patterns to be Ignored from Monitoring	By default, this test does not track requests to the following URL patterns: <i>*.js, *.css, *.jpeg, *.jpg, *.png</i>. 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 <i>.gif</i> and <i>.bmp</i> when monitoring, you need to alter the default specification as follows: <i>*.js, *.css, *.jpeg, *.jpg, *.png, *.gif, *.bmp</i> 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 <i>none</i> by default. Sometimes however, administrators may not want to be notified when certain

Parameter	Description
	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: <i><Javascript error message>:<URL of the page/file where the message originated></i>. For instance, say that the <i>login.html</i> page in your web site runs a few Java scripts that throw <i>Object expected errors</i>, which you want the eG agent to ignore. In this case, your error specification can be as follows: <i>Object expected:http://www.eazycart.com/web/login.html</i>. Alternatively, you can provide only that text string with which the error message begins in your specification – eg., <i>Object:http://www.eazycart.com/web/login.html</i>. 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: <i>Object:login.html</i>. 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 <i>login.html</i> page points to a <i>validate.js</i> file, and you want to ignore the <i>Object expected errors</i> that this JS file throws, your error pattern specification will either be, <i>Object expected:validate.js</i>, or <i>Object:validate.js</i>. Multiple error message-URL combinations can also be provided as a comma-separated list. The format of your specification will be: <i><Javascript error message 1>:<Originating URL 1>,<Javascript error message 2>:<Originating URL 2>,...</i> For example, to ignore the <i>Object expected</i> and <i>Uncaught TypeError</i> errors in the <i>login.html</i> page, use the following specification: <i>Object:login.html,Uncaught:login.html</i> Likewise, to ignore the <i>Object expected</i> error in the <i>login.html</i> page and the <i>Uncaught TypeError</i> in the <i>validate.js</i> file, your specification will be: <i>Object:login.html,Uncaught:validate.js</i> If you want to ignore the <i>Uncaught TypeError</i> across all the pages of the web site, your specification will be as follows: <i>Uncaught TypeError:All</i>

Parameter	Description
	Note: When specifying the <i><Javascript error message></i> 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 <i>case-sensitive</i> pattern matching.
Include Page Type Categorization in Top-N DD	By default, the detailed diagnosis of this test reports the top-n transactions (in terms of <i>Avg page load time</i>) across page types. Accordingly, the Include Page Type Categorization in Top-N DD flag is set to No by default. If you want detailed diagnosis to report the top-n transactions per page type, then set this flag to Yes. In this case, if say the Maximum Healthy Transactions, Maximum Slow Transactions, and Maximum Error Transactions parameters are each set to 5, then the detailed diagnosis of this test will report the top-5 healthy, top-5 slow, and top-5 error transactions to Ajax pages, iFrames, Virtual pages, and base pages. As this may typically generate a large volume of data and increase the load on the eG database, you may want to set the 'Include Page Type Categorization in Top-N DD' to 'Yes', only if your eG database is sized sufficiently.
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, for each measurement period in the timeline chosen for detailed diagnosis. To identify the top 5, the eG agent sorts all healthy transactions in a measurement period 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 <i>all</i>. Before setting the value of this parameter to <i>all</i>, make sure that you have a well-sized eG database in place, as this setting will store details of every transaction that registers an <i>Avg page load time</i> 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.

Parameter	Description
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 the top-5 slow transactions, in terms of the <i>Average page load time</i>, for each measurement period in the timeline chosen for detailed diagnosis. To identify the top 5, the eG agent sorts all slow transactions in a measurement period in the descending order of their <i>Average page load time</i>, 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 <i>0</i>. To view all slow transactions, set the value of this parameter to <i>all</i>. Before setting the value of this parameter to <i>all</i>, make sure that you have a well-sized eG database in place, as this setting will store details of every transaction that registers an <i>Avg page load time</i> 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 (per measurement period) that encountered JavaScript errors, based on when those errors occurred. To identify the top 5, the eG agent sorts all error transactions in a measurement period 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 <i>0</i>. To view all error transactions, set the value of this parameter to <i>all</i>. Before setting the value of this parameter to <i>all</i>, 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 <i>4000 millisecs</i> (i.e., 4 seconds). This means that, if a page takes more than 4 seconds to load, this test will consider

Parameter	Description
	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 <i>Maximum threshold</i> setting of the <i>Avg page load time</i> measure – i.e., both are set to <i>4000 millisecs</i> 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 <i>Avg page load time</i>. 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 Transaction Cutoff is changed to <i>6000 millisecs</i>, change the <i>Maximum Threshold</i> of the <i>Avg page load time</i> measure to <i>6000 millisecs</i> as well.
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, Virtual Page, and/or iFrame views as well in the dashboard, by selecting the relevant options from this list box.
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 <i>0</i> - 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 <i>0</i>. These zero values will clearly indicate to users that there is no traffic to the monitored web site/web application.
DD Frequency	Refers to the frequency with which detailed diagnosis measures are to be generated for this test. The default is <i>3:1</i> . This indicates that, by default, detailed measures will be generated every third 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 <i>none</i> 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.

Parameter	Description
	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 using this device.	Number	This is a good measure of the traffic from a specific device. 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 <i>Real User Monitor</i> 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

Measurement	Description	Measurement Unit	Interpretation
			unavailable. Because of this, the <i>Page views</i> 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 on this device.	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

Measurement	Description	Measurement Unit	Interpretation
			tolerating, then the Apdex score would be 0.50. Ideally therefore, the value of this measure should be <i>1.0</i>. A value less than <i>1.0</i> indicates that the experience of users from using this device has been less than satisfactory.
Average page load time	Indicates the average time that pages accessed using this device took to load completely on the device.	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 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 devices, you will be able to tell if the page load time is significantly higher for any one device – this could be the device that is causing the slowness. You may want to compare the values of the of the <i>Average browser initial request time</i>, <i>Average network time</i>, <i>Average server time</i>, <i>Average content download time</i>, and <i>Average browser render time</i> measures for that device to know what is exactly causing users using that device to experience slowness. If the <i>Average browser initial request time</i> is the highest, it can be attributed to a slowdown in redirection, AppCaching, or because the request spent too much time waiting on the device. If the <i>Average network time</i> is the highest, it denotes that the network is the problem source. This in turn can be caused by TCP connection latencies, delays in domain look up, and slowdown in SSL handshake.

Measurement	Description	Measurement Unit	Interpretation
			If the <i>Average content download time</i> measure reports an abnormally high value, then it could be owing to slowness in downloading the HTML document or in building the DOM. On the other hand, if the <i>Average server time</i> 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. To know which pages accessed using this device are slow, use the detailed diagnosis of this measure.
Unique user session	Indicates the number of distinct users who are currently accessing the web site/web application using this device.	Number	
Page views per minute	Indicates the number of times the pages were viewed per minute using this device.	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 on this device.	Percent	The value of this measure indicates the percentage of page views in which users using this device 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 on 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 view requests are either slow or have encountered Javascript errors on

Measurement	Description	Measurement Unit	Interpretation
			this device. Under such circumstances, to know what exactly is affecting the user experience on the device, compare the value of the <i>Slow page view percentage</i> with that of the <i>Javascript error page view percentage</i> for that device. 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 <i>Slow page views</i> measure to know which pages accessed by users using this device are slow and where these pages are losing time. If JavaScript errors are the problem, use the detailed diagnosis of the <i>JavaScript error view percentage</i> measure to know what errors occurred in which pages accessed using this device.
Slow page view percentage	Indicates the percentage of pages that were slow in loading when accessed using this device.	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 <i>Slow page views</i> 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 on this device.	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 on this device.	Number	A page view is considered to be slow when the average time taken to load that page exceeds the Slow Transaction

Measurement	Description	Measurement Unit	Interpretation
			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 using this device. To know which pages were slow when accessed using this device and why, use the detailed diagnosis of this measure.
JavaScript error page views	Indicates the number of times JavaScript errors occurred when accessing pages using this device.	Number	Ideally, the value of this measure should be 0. A high value indicates that many JavaScript errors are occurring when accessing pages using this device. 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 on 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 <i>Page views</i> 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 <i>Tolerating page views</i> and the <i>Frustrated page views</i>, it is a clear indicator that the experience of users on this device has been below-par.

Measurement	Description	Measurement Unit	Interpretation
			In such a case, use the detailed diagnosis of the <i>Tolerating page views</i> and <i>Frustrated page views</i> measures to know which pages are slow and why.
Tolerating page views	Indicates the number of tolerating page views experienced by users on this device.	Number	If the <i>Average page load time</i> of a page exceeds the Slow Transaction Cutoff configuration of this test, but is less than 4 times the Slow Transaction Cutoff (i.e., $< 4 * \text{Slow Transaction Cutoff}$), then such a page view is considered to be a <i>Tolerating page view</i>. Ideally, the value of this measure should be 0. A value higher than that of the <i>Satisfied page views</i> measure is a cause for concern, as it implies that the overall experience of users on this device is 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 on this device.	Number	If the <i>Average page load time</i> of a page is over 4 times the Slow Transaction Cutoff configuration of this test (i.e., $> 4 * \text{Slow Transaction Cutoff}$), 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 <i>Satisfied page views</i> measure is a cause for concern, as it implies that the experience of users has been less than satisfactory on this device. To know which pages are contributing to this sub-

Measurement	Description	Measurement Unit	Interpretation
			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 browser initial request time	Indicates the interval between when a request was received from users, when it followed a redirect, to when it was processed in the AppCache, on this device.	ms	In Figure 3.205, the <i>Average browser initial request time</i> is the time spent between the navigationStart event to the domainLookupStart event. This also includes the time spent by the browser waiting for one event to end and the next to begin. In short, this measure is the sum of <i>Average redirect time</i>, <i>Average AppCache time</i>, and <i>Average browser wait time</i> measures. This means that if a request takes too long to follow a redirection, or if the AppCache takes too long to process the request, or if the request spends too much time on the browser for the previous request to complete, then the value of this measure will significantly increase. This in turn will impact user experience, and consequently, the <i>Apdex score</i>. This is why, if this measure reports an abnormal value for a device, it is important that you compare the value of the <i>Average redirect time</i>, <i>Average AppCache time</i>, and <i>Average browser wait time</i> measures for the same device, to figure out where the request spent maximum time - in redirection? in the AppCache? or when waiting on the browser?
Average server time	Indicates the interval between the start of processing of a request	ms	In Figure 3.205, the <i>Average server time</i> is the time spent between the <i>requestStart</i> event and <i>responseStart</i>

Measurement	Description	Measurement Unit	Interpretation
	to when response is received on this device.		event. Ideally, a low value is desired for this measure, as high values will certainly hurt the <i>Apdex score</i> 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 <i>Average server time</i> will increase, leaving users with an unsatisfactory experience with the web site/web application. Note: This test uses the Navigation Timing API to measure web site performance. The Navigation Timing API typically exposes several properties that offer information about the time at which different page load events happen (see Figure 3.205) - eg., the <i>requestStart</i> event, the <i>responseStart</i> event, etc. This test uses the time stamps provided by the Navigation Timing API to compute and report the duration of page load events, so you can accurately identify where page loading is bottlenecked. The Navigation Timing API on Internet Explorer (IE) v11 reports an incorrect time stamp for the <i>requestStart</i> event of the

Measurement	Description	Measurement Unit	Interpretation
			page loading process (see Figure 3.205). As a result, for page view requests initiated from IE 11 browsers alone, eG RUM will report incorrect values for this measure. This issue was noticed in IE 11 in April 2019. It is recommended that you track hot fixes/patches released by Microsoft post April 2019, study the release notes of such fixes/patches, and determine if this bug has been fixed in any. If so, then you are advised to apply that fix/patch on IE 11 to resolve the issue. Until then, we recommend that you use the following workaround to accurately measure the <i>Average server time</i> of a page view request. - Deploy eG BTM (Java/.NET, as the case may be) on the backend server hosting the target web site/web application. - Use eG BTM to trace the path of transactions to the target web site/web application and enable the (Java or .NET) Business Transactions test to capture metrics on transaction performance. - Next, use the detailed diagnostics reported by eG RUM to identify the page view requests coming from the IE 11 browser. Make a note of the values in the Request time, URL and Query Params columns of detailed diagnosis. - Then, search the detailed

Measurement	Description	Measurement Unit	Interpretation
			diagnostics of the (Java or .NET) Business Transactions test for transactions with the same URL , Request time, and Query Params as reported by eG RUM. ◦ The response time that eG BTM reports for each of these transactions is the <i>Average server time</i> of those transactions. Note that this workaround applies only for those transaction URLs that are captured and reported as part of detailed diagnostics.
Average network time	Indicates the elapsed time since a user initiates a request to the web site/application and the start of fetching the response document from it, on this device.	ms	In Figure 3.205, the time spent between <i>navigationStart</i> and <i>requestStart</i> makes up the <i>Average network time</i>. This includes the time to perform DNS lookups, the time to establish a TCP connection with the server, and the time to perform an SSL handshake. In other words, the value of this measure is nothing but the sum of the <i>Average DNS lookup time</i> , the <i>Average TCP connection time</i> , and the <i>Average SSL handshake time</i> 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 <i>Average DNS lookup time</i> , <i>Average TCP connection time</i>, and <i>Average SSL handshake time</i> measures to know to what this delay can be attributed – a delay in domain name resolution? Or a poor network connection

Measurement	Description	Measurement Unit	Interpretation
			to the server? or slowness in SSL negotiations between the browser and the server? Note: This test uses the Navigation Timing API to measure web site performance. The Navigation Timing API typically exposes several properties that offer information about the time at which different page load events happen (see Figure 3.205) - eg., the <code>requestStart</code> event, the <code>responseStart</code> event, etc. This test uses the time stamps provided by the Navigation Timing API to compute and report the duration of page load events, so you can accurately identify where page loading is bottlenecked. The Navigation Timing API on Internet Explorer (IE) v11 reports an incorrect time stamp for the <i>requestStart</i> event of the page loading process (see 3.6.3). As a result, for page view requests initiated from IE 11 browsers alone, eG RUM will report incorrect values for this measure. This issue was noticed in IE 11 in April 2019. It is recommended that you track hot fixes/patches released by Microsoft post April 2019, study the release notes of such fixes/patches, and determine if this bug has been fixed in any. If so, then you are advised to apply that fix/patch on IE 11 to resolve the issue.
Average content download time	Indicates the time to make the complete HTML document (DOM) available for JavaScript to apply rendering logic on the pages accessed	ms	This is the time spent between the <code>responseStart</code> event (shown in 3.6.3) and the <code>documentContentLoadedEventStart</code> event (not shown in 3.6.3). The <code>documentContentLoadedEventStart</code> is

Measurement	Description	Measurement Unit	Interpretation
	using this device.		typically fired just before the <code>domContentLoaded</code> event, which is just after browser has finished downloading and parsing all the scripts that had <code>defer</code> set and no <code>async</code> attribute. In summary, the <i>Average content download time</i> measure is the sum of the values of the <i>Average DOM download time</i> and <i>Average DOM processing time</i> measures. This means that an abnormal increase in any of the above-mentioned time values will increase the value of this measure. If content downloading takes unusually long for a request, then you must compare the values of the <i>Average DOM download time</i> and <i>Average DOM processing time</i> measures to figure out what is causing the delay - is it because of the poor responsiveness of server, cache, or local resource? or is it because DOM processing took too long?
Average browser render time	Indicates the time taken to complete the download of remaining resources, including images, and to finish rendering the pages accessed using this device.	ms	A high value of this measure indicates that page rendering is taking too long. This can be attributed to an in-optimal HTML document architecture, complex CSS' (eg., deeply nested rules, slow selectors, complicated effects such as round borders, gradients etc.), and large sized images. If the <i>Average page load time</i> measure reports an abnormally high value, then you may want to compare the value of this measure with that of the <i>Average browser initial request time</i>, <i>Average network time</i>, <i>Average server time</i>, and

Measurement	Description	Measurement Unit	Interpretation
			<i>Average content download time</i> measures to nail the exact source of the bottleneck.
Average DOM download time	Indicates the time taken to download the complete HTML document for page view requests on this device.	ms	The value of this measure is the time that elapsed between the responseStart and responseEnd events in 3.6.3. 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 accessed using this device 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. Consequently, content download will be delayed, thus adversely impacting user experience with the web site/web application. Ideally therefore, the value of this measure should be low.
Average browser wait time	Indicates the time that requests from this device spent on the browser, waiting for another request to complete.	ms	This is the sum of the time between every two consecutive events, starting with the navigationStart event till the requestStart event in 3.6.3. Typically, web browsers limit the number of active connections for each domain. Most modern browsers (eg., Chrome) support only six simultaneous requests/connections. In this case

Measurement	Description	Measurement Unit	Interpretation
			therefore, when the seventh request comes in, that request waits on the browser until the six requests sent previously are processed. The waiting time of the seventh request is the browser wait time. High browser wait time can prolong the browser's initial request time, thus adversely impacting the overall responsiveness of the web site/application. This is why, if the <i>Average browser initial request time</i> measure reports an abnormally high value, you will have to compare the values of the <i>Average browser wait time</i>, <i>Average redirection time</i>, and <i>Average AppCache time</i> measure to determine whether/not the initial request delay observed on the browser is because requests have been waiting on the browser for too long a time. Some of the means by which you can reduce browser waits are briefly discussed below: • Browsers such as Mozilla Firefox support upto 10 parallel requests. You may want to recommend such browsers for your web site/web application users, so that more requests are processed and fewer requests are queued on the browser, thus reducing browser wait time. • Web site/application developers can try domain sharding - i.e., split content

Measurement	Description	Measurement Unit	Interpretation
			across multiple domains. Typically, when a user connects to a web page, his or her browser scans the resulting HTML for resources to download. Normally these resources are supplied by a single domain - the domain providing the web page or a domain created specifically for resources. With domain sharding, the user's browser connects to two or more different domains to simultaneously download the resources needed to render the web page. This allows website/application to be delivered faster to users as they do not have to wait for the previous set of requests to end before beginning the next set.
Average redirection time	Indicates the time that requests received from this device spent in redirection before fetching the pages.	ms	In 3.6.3, this is the elapsed time between the redirectStart and redirectEnd events. URL redirection, also known as URL forwarding, is a technique to give a page, a form, or a whole web site/application, more than one URL address. Usually, web site administrators use URL redirection to: • Redirect users to the mobile version of the site • Redirect users to secured pages • Redirect users to the latest version of the resource/content • Redirect users to pages specific to

Measurement	Description	Measurement Unit	Interpretation
			their geo location • Redirecting canonical URLs Though redirects are useful, they have to be kept at a minimum, as each redirect on a page adds latency to the overall page load time. This is because, when a user enters a domain into the browser and hits enter, the DNS resolution process is triggered and the domain is resolved to its corresponding IP address in a few milliseconds. If the landing page has another redirect, then the browser repeats the entire DNS resolution process once again to guide the user to the correct web page. The multiple redirect requests are taxing on the browser resources and slow down the page load. Web page load time is also affected by internal redirects; for example, if the page tries to load content from a URL that has been redirected to newer or updated content, then the browser must create additional requests to fetch the valid content. These redirects result in additional round trips between the browser and the web server which pushes the load time higher; the perceived performance is degraded every time the browser encounters a redundant redirect. Web site/application performance is also impacted if redirects are not implemented correctly. Some of the common redirect errors are: • Multiple redirects: The higher the number of redirects on a page, the higher is its page load time.

Measurement	Description	Measurement Unit	Interpretation
			- Invalid redirects: There are often instances where the web site administrator sets up bulk redirects without verifying the validity of the redirects. The site may also have old redirects that were never cleaned up. This can cause several issues on the site like broken links and 404s. - Redirect loop: When there are several redirects on the page that are linked to each other, it creates a chain of redirects which may loop back to the same URL that initiated the redirect. This results in a redirect loop error and the user will not be able to access the site. Therefore, if you find that the value of the <i>Average page load time</i> measure is abnormally high owing to an unusually high value for the <i>Average redirection time</i> measure, then make sure you follow the best practices outlined below while implementing redirects, so you can significantly reduce page load time and improve user experience: - Avoid redundant redirects: It's recommended to avoid redirects where possible and to use this method only when absolutely needed. This will cut down unnecessary overhead and improve the perceived performance of the page.

Measurement	Description	Measurement Unit	Interpretation
			- Chain redirects: When a URL is linked to another URL, this creates a chained redirect. Each URL added to the chain adds latency to the page. Chained redirects have a negative impact not only on page speed, but also SEO. - Clean-up redirects: You may have hundreds of redirects on your website and it could be one of the main factors affecting page speed. Old redirects may conflict with new URLs, backlinks can cause odd errors on the page. It is recommended to verify all the redirects you have set up and to remove the ones that are no longer needed. Retain the old links that have major referral traffic, while those that are rarely accessed can be removed. This exercise will help improve page speed significantly.
Average AppCache time	Indicates the time taken to check whether/not the requests received from this device can be serviced by the AppCache.	ms	HTML5 provides an application caching mechanism that lets web-based applications run offline. Developers can use the Application Cache (AppCache) interface to specify resources that the browser should cache and make available to offline users. Applications that are cached load and work correctly even if users click the refresh button when they are offline. Using an application cache gives an application the following benefits: - Offline browsing: users can navigate a site even when they are offline. - Speed: cached resources are local, and

Measurement	Description	Measurement Unit	Interpretation
			therefore load faster. Reduced server load: the browser only downloads resources that have changed from the server. To enable the application cache for an application, you must include the manifest attribute in the <html> element in your application's pages. The manifest attribute references a cache manifest file, which is a text file that lists resources (files) that the browser should cache for your application. The browser does not cache pages that do not contain the manifest attribute, unless such pages are explicitly listed in the manifest file itself. You do not need to list all the pages you want cached in the manifest file; the browser implicitly adds every page that the user visits and that has the manifest attribute set to the application cache. When the browser visits a document that includes the manifest attribute, if no application cache exists, the browser loads the document and then fetches all the entries listed in the manifest file, creating the first version of the application cache. Subsequent visits to that document cause the browser to load the document and other assets specified in the manifest file from the application cache. If the manifest file has changed, all the files listed in the manifest—as well as those already added to the cache—are fetched into a temporary cache.

Measurement	Description	Measurement Unit	Interpretation
			Once all the files have been successfully retrieved, they are moved into the real offline cache automatically. Since the document has already been loaded into the browser from the cache, the updated document will not be rendered until the document is reloaded (either manually or programmatically). A high value of this measure signifies that requests are spending too much time in the AppCache. This also introduces page loading latencies, which have an adverse effect on user-perceived performance of a web site/application. Common reasons for AppCaching issues and their practical solutions are detailed below: • If the media type is not set, then AppCache will not work. To avoid this, make sure that the manifest file is always served under the correct media type of text/cache-manifest. • If the manifest file is not served to the web browser from the same origin as the host page, the manifest file will fail to load. To avoid this, make sure that the manifest file is always served from the same origin as the host page. However, note that the manifest file can hold reference to resources to be cached from other domains. • The relative URLs that you mention in the manifest are relative to the manifest file and not to the document where you

Measurement	Description	Measurement Unit	Interpretation
			reference the manifest file. If you make this error when the manifest and the reference are not in the same path, the resources will fail to load and in turn the manifest file will not be loaded. This will stall appcaching. Any change made to the manifest file will cause the entire set of files to be downloaded again. Moreover, if a manifest file is added to a HTML file, it forces all resources to be downloaded synchronously as soon as the manifest file is downloaded. As a result, resources that may not yet be required, such as JavaScript or an image below the fold, will be downloaded at the start of the page. This can increase page load time significantly. The solution to this is to load the Application Cache from a simple HTML file loaded in an iframe. This not only avoids caching dynamic HTML but also avoids the Application Cache being downloaded asynchronously after the page load has completed.
Average DNS lookup time	Indicates the time taken by the browser to perform the domain lookup for connecting to the pages requested from this device.	ms	A high value for this measure will not only affect DNS lookup, but will also impact the <i>Average network time</i> and <i>Average page load time</i> 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 for accessing pages requested from	ms	A bad network connection between the browser client and the server can delay TCP connections to the server. As a result, the <i>Average network time</i> too will increase, thus impacting page load time and overall user experience with the web

Measurement	Description	Measurement Unit	Interpretation
	this device.		site/web application.
Average SSL handshake time	Indicates the time taken by requests to complete the SSL handshake on this device.	ms	An SSL handshake happens when a browser makes a secure request for content, also known as an encrypted HTTPS connection. The user's browser and server negotiate encrypted keys and certificates to establish a secure connection between each other. Because this SSL negotiation requires exchanges between the browser and your server, it increases the time spent by the request on the network (i.e., it adds to the value of the Average network time measure). This in turn increases page load time. In fact, an SSL handshake, along with DNS lookup and TCP handshake add three round trips to the page load time. A quick fix to this is to use HTTP/2. HTTP/2 can use caching to reduce SSL setup to only one round trip.

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:** iFrames 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 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

Configurable parameters for the test

Parameter	Description
Test Period	How often should the test be executed.
Proxy Host	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 <i>none</i> , indicating that the eG agent does not communicate with the collector via a proxy.
Proxy Port	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 <i>none</i> , indicating that the eG agent does not communicate with the collector via a proxy.
Proxy Username and Password	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 Proxy Username and Proxy Password . If no proxy server is used or if the proxy

Parameter	Description
	server used does not require authentication, then set the ProxyUsername and ProxyPassword to <i>none</i> .
Confirm Password	Confirm the Proxy Password by retyping it here. Note: If you Reconfigure the test later to change the values of the Proxy Username, Proxy Password, 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. MAXIMUM ALLOWED PAGE VISITS PER DAY 100000 Configuring the number of allowed page visits By default, the Maximum allowed page visits per day is set to <i>100000</i>. This implies that the test will consider only the first <i>100000</i> requests in a day for monitoring. All page visits beyond <i>100000</i> will by default be excluded from the test's monitoring purview. You can increase or decrease this limit, if you so need.
Consider JavaScript errors in Apdex calculations	The formula for computing the Apdex score is as follows: $[Satisfied\ requests + (Tolerating\ requests/2)]/Total\ samples$ By default, the count of <i>Satisfied</i> and <i>Tolerating requests</i> used for Apdex score computation includes the count of requests that experienced JavaScript errors. This is why, the Consider JavaScript errors in Apdex calculations flag is set to Yes by default. Since JavaScript errors also impact user experience, this default setting will result in an Apdex score that presents a near-real picture of the performance of a web site/web application. If you do not want the Apdex score to be tainted by error requests, then set this flag to <i>No</i>. In this case, requests with JavaScript errors will not be considered for Apdex calculations, even if such requests have a response time that is well-within (satisfied) or in violation (tolerating) of the Slow Transaction Cutoff.
URL Segments to be used as Grouped	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

Parameter	Description
URL	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.eazycart.com/web/shopping/login.jsp, then http://www.eazycart.com will be the base URL or domain, <i>/web</i> will be the first URL segment, <i>/shopping</i> will be the second URL segment, and <i>/login.jsp</i> 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 <i>/web/shopping</i>. Here, <i>/web</i> corresponds to the specification 1 (URL segment 1) and <i>/shopping</i> 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.eazycart.com/web/shopping/products.jsp, http://www.eazycart.com/web/shopping/products/travel/bags.jsp, http://www.eazycart.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 <i>/web/shopping</i>. This way, the test will create different page groups based on each of the second-level URL segments in the managed web site – eg., <i>/web/movies</i>, <i>/web/travel</i>, <i>/web/reservations</i>, <i>/partner/contacts</i> 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.eazycart.com/web/shopping/login.jsp, the test will report metrics for the descriptor <i>/shopping</i>. You can even set this parameter to 1,3. For a web site that contains URLs such as http://www.eazycart.com/web/shopping/products.jsp, http://www.eazycart.com/web/shopping/products/travel/bags.jsp, and http://www.eazycart.com/web/shopping/payment.jsp, this specification will result in the following descriptors: <i>/web/products</i>, <i>/web/products.jsp</i>, and <i>/web/payment.jsp</i>.
URL Patterns to be Ignored from Monitoring	By default, this test does not track requests to the following URL patterns: <i>*.js, *.css, *.jpeg, *.jpg, *.png</i>. 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 <i>.gif</i> and <i>.bmp</i> when monitoring, you need to alter the default specification as follows: <i>*.js, *.css, *.jpeg, *.jpg, *.png, *.gif, *.bmp</i> Note:

Parameter	Description
	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 <i>none</i> 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: <i><Javascript error message>:<URL of the page/file where the message originated></i>. For instance, say that the <i>login.html</i> page in your web site runs a few Java scripts that throw <i>Object expected errors</i>, which you want the eG agent to ignore. In this case, your error specification can be as follows: <i>Object expected:http://www.eazycart.com/web/login.html</i>. Alternatively, you can provide only that text string with which the error message begins in your specification – eg., <i>Object:http://www.eazycart.com/web/login.html</i>. 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: <i>Object:login.html</i>. 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

Parameter	Description
	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 <i>login.html</i> page points to a <i>validate.js</i> file, and you want to ignore the <i>Object expected</i> errors that this JS file throws, your error pattern specification will either be, <i>Object expected:validate.js</i>, or <i>Object:validate.js</i>. Multiple error message-URL combinations can also be provided as a comma-separated list. The format of your specification will be: <i><Javascript error message 1>:<Originating URL 1>,<Javascript error message 2>:<Originating URL 2>,. . .</i> For example, to ignore the <i>Object expected</i> and <i>Uncaught TypeError</i> errors in the <i>login.html</i> page, use the following specification: <i>Object:login.html,Uncaught:login.html</i> Likewise, to ignore the <i>Object expected</i> error in the <i>login.html</i> page and the <i>Uncaught TypeError</i> in the <i>validate.js</i> file, your specification will be: <i>Object:login.html,Uncaught:validate.js</i> If you want to ignore the <i>Uncaught TypeError</i> across all the pages of the web site, your specification will be as follows: <i>Uncaught TypeError:All</i> Note: When specifying the <i><Javascript error message></i> 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 <i>case-sensitive</i> pattern matching.
Include Page Type Categorization in Top-N DD	By default, the detailed diagnosis of this test reports the top-n transactions (in terms of <i>Avg page load time</i>) across page types. Accordingly, the Include Page Type Categorization in Top-N DD flag is set to No by default. If you want detailed diagnosis to report the top-n transactions per page type, then set this flag to Yes. In this case, if say the Maximum Healthy Transactions, Maximum Slow Transactions, and Maximum Error Transactions parameters are each set to 5, then the detailed diagnosis of this test will report the top-5 healthy, top-5 slow, and top-5 error

Parameter	Description
	transactions to Ajax pages, iFrames, Virtual pages, and base pages. As this may typically generate a large volume of data and increase the load on the eG database, you may want to set the 'Include Page Type Categorization in Top-N DD' to 'Yes', only if your eG database is sized sufficiently.
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, for each measurement period in the timeline chosen for detailed diagnosis. To identify the top 5, the eG agent sorts all healthy transactions in a measurement period 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 <i>all</i>. Before setting the value of this parameter to <i>all</i>, make sure that you have a well-sized eG database in place, as this setting will store details of every transaction that registers an <i>Avg page load time</i> 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 the top-5 slow transactions, in terms of the <i>Average page load time</i>, for each measurement period in the timeline chosen for detailed diagnosis. To identify the top 5, the eG agent sorts all slow transactions in a measurement period in the descending order of their <i>Average page load time</i>, 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 <i>all</i>. Before setting the value of this parameter to <i>all</i>, make sure that you have a well-sized eG database in place, as this setting will store details of every transaction that registers an <i>Avg page load time</i> value that is higher than the slow transaction cutoff specification. On a bad day, this can result in numerous transactions, and can

Parameter	Description
	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 (per measurement period) that encountered JavaScript errors, based on when those errors occurred. To identify the top 5, the eG agent sorts all error transactions in a measurement period 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 <i>Maximum threshold</i> setting of the <i>Avg page load time</i> 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 <i>Avg page load time</i>. 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 Transaction Cutoff is changed to 6000 millisecs, change the <i>Maximum Threshold</i> of the <i>Avg page load time</i> measure to 6000 millisecs as well.
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, Virtual Page, and/or iFrame views as well in the dashboard, by selecting the relevant options from this list box.

Parameter	Description
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 is no traffic to the monitored web site/web application.
DD Frequency	Refers to the frequency with which detailed diagnosis measures are to be generated for this test. The default is 3:1. This indicates that, by default, detailed measures will be generated every third 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 <i>none</i> 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,

Measurement	Measurement	Measurement Unit	Interpretation
			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 <i>Real User Monitor</i> 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 <i>Page views</i> 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 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.

Measurement	Measurement	Measurement Unit	Interpretation
			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 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.

Measurement	Measurement	Measurement Unit	Interpretation
			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 <i>Average browser initial request time</i>, <i>Average network time</i>, <i>Average server time</i>, <i>Average content download time</i>, and <i>Average browser render time</i> measures for that page type, to know what is exactly causing pages of that type to load slowly. If the <i>Average browser initial request time</i> is the highest, it can be attributed to a slowdown in redirection, AppCaching, or because the request spent too much time waiting on the browser. If the <i>Average network time</i> is the highest, it denotes that the network is the problem source. This in turn can be caused by TCP connection latencies, delays in domain look up, and slowdown in SSL handshake. If the <i>Average content download time</i> measure reports an abnormally high value, then it could be owing to slowness in downloading the HTML document or in building the DOM. On the other hand, if the <i>Average server time</i> 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. To know which pages of the type are slow, use the detailed diagnosis of this measure.

Measurement	Measurement	Measurement Unit	Interpretation
Unique user session	Indicates the number of distinct users who are currently accessing pages of this type in the web site/web application.	Number	HttpOnly is a flag that can be set on cookies to prevent their contents from being accessed by JavaScript. This is often done for session cookies to hide the session identifier, as a security measure. If the HttpOnly flag is set for the session cookies of the monitored web site, then the eG RUM JavaScript will not be able to read the session identifier from the cookies. In such cases therefore, the eG agent will not be able to report the count of unique sessions.
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 <i>Slow page view percentage</i> with that of the <i>Javascript error page view percentage</i>

Measurement	Measurement	Measurement Unit	Interpretation
			for that page type. 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 <i>Slow page views</i> measure to know which pages of that type are slow and where these pages are losing time. If JavaScript errors are the problem, use the detailed diagnosis of the <i>JavaScript error view percentage</i> measure to know what errors occurred in which pages of the type.
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 <i>Slow page views</i> 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

Measurement	Measurement	Measurement Unit	Interpretation
			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!
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 <i>Page views</i> 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 <i>Tolerating page views</i> and the <i>Frustrated page views</i>, it is a clear indicator that the user experience with this page type has been below-par. In such a case, use the detailed diagnosis of the <i>Tolerating page views</i> and <i>Frustrated page views</i> 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 <i>Average page load time</i> of a page exceeds the Slow Transaction Cutoff configuration of this test, but is less than 4

Measurement	Measurement	Measurement Unit	Interpretation
			times the Slow Transaction Cutoff (i.e., $< 4 * \text{Slow Transaction Cutoff}$), then such a page view is considered to be a <i>Tolerating page view</i>. Ideally, the value of this measure should be 0. A value higher than that of the <i>Satisfied page views</i> measure is a cause for concern, as it implies that the overall 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 <i>Average page load time</i> of a page is over 4 times the Slow Transaction Cutoff configuration of this test (i.e., $> 4 * \text{Slow Transaction Cutoff}$), 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 <i>Satisfied page views</i> 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 browser	Indicates the interval	ms	In Figure 3.205, the <i>Average browser</i>

Measurement	Measurement	Measurement Unit	Interpretation
initial request time	between when a request was received by pages of this type, when it followed a redirect, to when it was processed in the AppCache.		<i>initial request time</i> is the time spent between the <i>navigationStart</i> event to the <i>domainLookupStart</i> event. This also includes the time spent by the browser waiting for one event to end and the next to begin. In short, this measure is the sum of <i>Average redirect time</i>, <i>Average AppCache time</i>, and <i>Average browser wait time</i> measures. This means that if a request takes too long to follow a redirection, or if the AppCache takes too long to process the request, or if the request spends too much time on the browser for the previous request to complete, then the value of this measure will significantly increase. This in turn will impact user experience, and consequently, the <i>Apdex score</i>. This is why, if this measure reports an abnormal value for a page type, it is important that you compare the value of the <i>Average redirect time</i>, <i>Average AppCache time</i>, and <i>Average browser wait time</i> measures for the same page type, to figure out where the request spent maximum time - in redirection? in the AppCache? or when waiting on the browser?
Average server 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 3.205, the <i>Average server time</i> is the time spent between the <i>requestStart</i> event and <i>responseStart</i> event. Ideally, a low value is desired for this measure, as high values will certainly hurt the <i>Apdex score</i> of the web site/web application. The key factor that can influence the value of this measure is the request processing

Measurement	Measurement	Measurement Unit	Interpretation
			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 <i>Average server time</i> will increase, leaving users with an unsatisfactory experience with the web site/web application. Note: This test uses the Navigation Timing API to measure web site performance. The Navigation Timing API typically exposes several properties that offer information about the time at which different page load events happen (see Figure 3.205) - eg., the <i>requestStart</i> event, the <i>responseStart</i> event, etc. This test uses the time stamps provided by the Navigation Timing API to compute and report the duration of page load events, so you can accurately identify where page loading is bottlenecked. The Navigation Timing API on Internet Explorer (IE) v11 reports an incorrect time stamp for the <i>requestStart</i> event of the page loading process (see Figure 3.205). As a result, for page view requests initiated from IE 11 browsers alone, eG RUM will report incorrect values for this measure. This issue was noticed in IE 11 in April 2019. It is recommended that you track

Measurement	Measurement	Measurement Unit	Interpretation
			hot fixes/patches released by Microsoft post April 2019, study the release notes of such fixes/patches, and determine if this bug has been fixed in any. If so, then you are advised to apply that fix/patch on IE 11 to resolve the issue. Until then, we recommend that you use the following workaround to accurately measure the <i>Average server time</i> of a page view request. - Deploy eG BTM (Java/.NET, as the case may be) on the backend server hosting the target web site/web application. - Use eG BTM to trace the path of transactions to the target web site/web application and enable the (Java or .NET) Business Transactions test to capture metrics on transaction performance. - Next, use the detailed diagnostics reported by eG RUM to identify the page view requests coming from the IE 11 browser. Make a note of the values in the Request time, URL and Query Params columns of detailed diagnosis. - Then, search the detailed diagnostics of the (Java or .NET) Business Transactions test for transactions with the same URL, Request time, and Query Params as reported by eG RUM. - The response time that

Measurement	Measurement	Measurement Unit	Interpretation
			eG BTM reports for each of these transactions is the <i>Average server time</i> of those transactions. Note that this workaround applies only for those transaction URLs that are captured and reported as part of detailed diagnostics.
Average network 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 3.205, the time spent between <i>navigationStart</i> and <i>requestStart</i> makes up the <i>Average network time</i>. This includes the time to perform DNS lookups, the time to establish a TCP connection with the server, and the time to perform an SSL handshake. In other words, the value of this measure is nothing but the sum of the <i>Average DNS lookup time</i> , the <i>Average TCP connection time</i> , and the <i>Average SSL handshake time</i> 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 <i>Average DNS lookup time</i> , <i>Average TCP connection time</i>, and <i>Average SSL handshake time</i> measures to know to what this delay can be attributed – a delay in domain name resolution? Or a poor network connection to the server? or slowness in SSL negotiations between the browser and the server? Note: This test uses the Navigation Timing

Measurement	Measurement	Measurement Unit	Interpretation
			API to measure web site performance. The Navigation Timing API typically exposes several properties that offer information about the time at which different page load events happen (see Figure 3.205) - eg., the <code>requestStart</code> event, the <code>responseStart</code> event, etc. This test uses the time stamps provided by the Navigation Timing API to compute and report the duration of page load events, so you can accurately identify where page loading is bottlenecked. The Navigation Timing API on Internet Explorer (IE) v11 reports an incorrect time stamp for the <i>requestStart</i> event of the page loading process (see 3.6.4). As a result, for page view requests initiated from IE 11 browsers alone, eG RUM will report incorrect values for this measure. This issue was noticed in IE 11 in April 2019. It is recommended that you track hot fixes/patches released by Microsoft post April 2019, study the release notes of such fixes/patches, and determine if this bug has been fixed in any. If so, then you are advised to apply that fix/patch on IE 11 to resolve the issue.
Average content download time	Indicates the time to make the complete HTML document (DOM) available for JavaScript to apply rendering logic on the pages of this type.	ms	This is the time spent between the <code>responseStart</code> event (shown in 3.6.4) and the <code>documentContentLoadedEventStart</code> event (not shown in 3.6.4). The <code>documentContentLoadedEventStart</code> is typically fired just before the <code>domContentLoaded</code> event, which is just after browser has finished downloading and parsing all the scripts that had <code>defer</code> set and no <code>async</code> attribute. In summary, the <i>Average content</i>

Measurement	Measurement	Measurement Unit	Interpretation
			<i>download time</i> measure is the sum of the values of the <i>Average DOM download time</i> and <i>Average DOM processing time</i> measures. This means that an abnormal increase in any of the above-mentioned time values will increase the value of this measure. If content downloading takes unusually long for a request, then you must compare the values of the <i>Average DOM download time</i> and <i>Average DOM processing time</i> measures to figure out what is causing the delay - is it because of the poor responsiveness of server, cache, or local resource? or is it because DOM processing took too long?
Average browser render time	Indicates the time taken to complete the download of remaining resources, including images, and to finish rendering the pages of this type.	ms	A high value of this measure indicates that page rendering is taking too long. This can be attributed to an in-optimal HTML document architecture, complex CSS' (eg., deeply nested rules, slow selectors, complicated effects such as round borders, gradients etc.), and large sized images. If the <i>Average page load time</i> measure reports an abnormally high value, then you may want to compare the value of this measure with that of the <i>Average browser initial request time</i>, <i>Average network time</i>, <i>Average server time</i>, and <i>Average content download time</i> measures to nail the exact source of the bottleneck.
Average DOM download time	Indicates the time for the browser to download the complete HTML	ms	The value of this measure is the time that elapsed between the responseStart and responseEnd events in 3.6.4.

Measurement	Measurement	Measurement Unit	Interpretation
	document content to the pages of this type.		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 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. Consequently, content download will be delayed, thus adversely impacting user experience with the web site/web application. Ideally therefore, the value of this measure should be low.
Average browser wait time	Indicates the time that requests to this page type spent on the browser, waiting for another request to complete.	ms	This is the sum of the time between every two consecutive events, starting with the navigationStart event till the requestStart event in 3.6.4. Typically, web browsers limit the number of active connections for each domain. Most modern browsers (eg., Chrome) support only six simultaneous requests/connections. In this case therefore, when the seventh request comes in, that request waits on the browser until the six requests sent previously are processed. The waiting time of the seventh request is the browser wait time. High browser wait time can prolong the browser's initial request time, thus adversely impacting the overall responsiveness of the web site/application. This is why, if the <i>Average browser initial request time</i>

Measurement	Measurement	Measurement Unit	Interpretation
			measure reports an abnormally high value, you will have to compare the values of the <i>Average browser wait time</i>, <i>Average redirection time</i>, and <i>Average AppCache time</i> measure to determine whether/not the initial request delay observed on the browser is because requests have been waiting on the browser for too long a time. Some of the means by which you can reduce browser waits are briefly discussed below: • Browsers such as Mozilla Firefox support upto 10 parallel requests. You may want to recommend such browsers for your web site/web application users, so that more requests are processed and fewer requests are queued on the browser, thus reducing browser wait time. • Web site/application developers can try domain sharding - i.e., split content across multiple domains. Typically, when a user connects to a web page, his or her browser scans the resulting HTML for resources to download. Normally these resources are supplied by a single domain - the domain providing the web page or a domain created specifically for resources. With domain sharding, the user's browser connects to two or more different

Measurement	Measurement	Measurement Unit	Interpretation
			domains to simultaneously download the resources needed to render the web page. This allows website/application to be delivered faster to users as they do not have to wait for the previous set of requests to end before beginning the next set.
Average redirection time	Indicates the time that requests to pages of this type spent in redirection before fetching the pages.	ms	In 3.6.4, this is the elapsed time between the redirectStart and redirectEnd events. URL redirection, also known as URL forwarding, is a technique to give a page, a form, or a whole web site/application, more than one URL address. Usually, web site administrators use URL redirection to: • Redirect users to the mobile version of the site • Redirect users to secured pages • Redirect users to the latest version of the resource/content • Redirect users to pages specific to their geo location • Redirecting canonical URLs Though redirects are useful, they have to be kept at a minimum, as each redirect on a page adds latency to the overall page load time. This is because, when a user enters a domain into the browser and hits enter, the DNS resolution process is triggered and the domain is resolved to its corresponding IP address in a few milliseconds. If the landing page has another redirect, then the browser repeats

Measurement	Measurement	Measurement Unit	Interpretation
			the entire DNS resolution process once again to guide the user to the correct web page. The multiple redirect requests are taxing on the browser resources and slow down the page load. Web page load time is also affected by internal redirects; for example, if the page tries to load content from a URL that has been redirected to newer or updated content, then the browser must create additional requests to fetch the valid content. These redirects result in additional round trips between the browser and the web server which pushes the load time higher; the perceived performance is degraded every time the browser encounters a redundant redirect. Web site/application performance is also impacted if redirects are not implemented correctly. Some of the common redirect errors are: • Multiple redirects: The higher the number of redirects on a page, the higher is its page load time. • Invalid redirects: There are often instances where the web site administrator sets up bulk redirects without verifying the validity of the redirects. The site may also have old redirects that were never cleaned up. This can cause several issues on the site like broken links and 404s. • Redirect loop: When there are several redirects on the page that are linked to

Measurement	Measurement	Measurement Unit	Interpretation
			each other, it creates a chain of redirects which may loop back to the same URL that initiated the redirect. This results in a redirect loop error and the user will not be able to access the site. Therefore, if you find that the value of the <i>Average page load time</i> measure is abnormally high owing to an unusually high value for the Average redirection time measure, then make sure you follow the best practices outlined below while implementing redirects, so you can significantly reduce page load time and improve user experience: • Avoid redundant redirects: It's recommended to avoid redirects where possible and to use this method only when absolutely needed. This will cut down unnecessary overhead and improve the perceived performance of the page. • Chain redirects: When a URL is linked to another URL, this creates a chained redirect. Each URL added to the chain adds latency to the page. Chained redirects have a negative impact not only on page speed, but also SEO. • Clean-up redirects: You may have hundreds of redirects on your website and it could be one of the main factors affecting page speed. Old redirects

Measurement	Measurement	Measurement Unit	Interpretation
			may conflict with new URLs, backlinks can cause odd errors on the page. It is recommended to verify all the redirects you have set up and to remove the ones that are no longer needed. Retain the old links that have major referral traffic, while those that are rarely accessed can be removed. This exercise will help improve page speed significantly.
Average AppCache time	Indicates the time taken to check whether/not the requests to this page type can be serviced by the AppCache.	ms	HTML5 provides an application caching mechanism that lets web-based applications run offline. Developers can use the Application Cache (AppCache) interface to specify resources that the browser should cache and make available to offline users. Applications that are cached load and work correctly even if users click the refresh button when they are offline. Using an application cache gives an application the following benefits: • Offline browsing: users can navigate a site even when they are offline. • Speed: cached resources are local, and therefore load faster. • Reduced server load: the browser only downloads resources that have changed from the server. To enable the application cache for an application, you must include the manifest attribute in the <html> element in your application's pages. The manifest attribute references a cache manifest file, which is a text file that lists resources (files) that the browser should cache for your

Measurement	Measurement	Measurement Unit	Interpretation
			application. The browser does not cache pages that do not contain the manifest attribute, unless such pages are explicitly listed in the manifest file itself. You do not need to list all the pages you want cached in the manifest file, the browser implicitly adds every page that the user visits and that has the manifest attribute set to the application cache. When the browser visits a document that includes the manifest attribute, if no application cache exists, the browser loads the document and then fetches all the entries listed in the manifest file, creating the first version of the application cache. Subsequent visits to that document cause the browser to load the document and other assets specified in the manifest file from the application cache. If the manifest file has changed, all the files listed in the manifest—as well as those already added to the cache—are fetched into a temporary cache. Once all the files have been successfully retrieved, they are moved into the real offline cache automatically. Since the document has already been loaded into the browser from the cache, the updated document will not be rendered until the document is reloaded (either manually or programmatically). A high value of this measure signifies that requests are spending too much time in the AppCache. This also introduces page loading latencies, which have an adverse effect on user-perceived performance of a

Measurement	Measurement	Measurement Unit	Interpretation
			web site/application. Common reasons for AppCaching issues and their practical solutions are detailed below: • If the media type is not set, then AppCache will not work. To avoid this, make sure that the manifest file is always served under the correct media type of text/cache-manifest. • If the manifest file is not served to the web browser from the same origin as the host page, the manifest file will fail to load. To avoid this, make sure that the manifest file is always served from the same origin as the host page. However, note that the manifest file can hold reference to resources to be cached from other domains. • The relative URLs that you mention in the manifest are relative to the manifest file and not to the document where you reference the manifest file. If you make this error when the manifest and the reference are not in the same path, the resources will fail to load and in turn the manifest file will not be loaded. This will stall appcaching. • Any change made to the manifest file will cause the entire set of files to be downloaded again. Moreover, if a manifest file is added to a HTML file, it forces all resources to be downloaded

Measurement	Measurement	Measurement Unit	Interpretation
			synchronously as soon as the manifest file is downloaded. As a result, resources that may not yet be required, such as JavaScript or an image below the fold, will be downloaded at the start of the page. This can increase page load time significantly. The solution to this is to load the Application Cache from a simple HTML file loaded in an iframe. This not only avoids caching dynamic HTML but also avoids the Application Cache being downloaded asynchronously after the page load has completed.
Average DNS lookup time	Indicates the time taken by this requests to 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 <i>Average network time</i> and <i>Average page load time</i> 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 requests to this page type 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 <i>Average network time</i> too will increase, thus impacting page load time and overall user experience with the web site/web application.
Average SSL handshake time	Indicates the time taken by requests to this page type to complete the SSL handshake.	ms	An SSL handshake happens when a browser makes a secure request for content, also known as an encrypted HTTPS connection. The user's browser and server negotiate encrypted keys and certificates to establish a secure connection between each other. Because this SSL negotiation requires exchanges between the browser and your server, it increases the time spent by the request on the network (i.e., it adds to the value of the

Measurement	Measurement	Measurement Unit	Interpretation
			Average network time measure). This in turn increases page load time. In fact, an SSL handshake, along with DNS lookup and TCP handshake add three round trips to the page load time. A quick fix to this is to use HTTP/2. HTTP/2 can use caching to reduce SSL setup to only one round trip.

3.6.5 Page Groups Test

The **Page Groups** test groups URLs based on the URL segments configured for monitoring and reports aggregated response 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.eazycart.com/web/shopping/login.jsp>, then <http://www.eazycart.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

Configurable parameters for the test

Parameter	Description
Test Period	How often should the test be executed.
Proxy Host	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 <i>none</i> , indicating that the eG agent does not communicate with the collector via a proxy.
Proxy Port	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 <i>none</i> , indicating that the eG agent does not communicate with the collector via a proxy.
Proxy Username and Proxy Password	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 Proxy Username and Proxy Password . If no proxy server is used or if the proxy server used does not require authentication, then set the ProxyUsername and ProxyPassword to <i>none</i> .
Confirm Password	Confirm the Proxy Password by retyping it here. Note: If you Reconfigure the test later to change the values of the Proxy Username , Proxy Password , 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.

MAXIMUM ALLOWED PAGE VISITS PER DAY

100000

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.

Parameter	Description
Consider JavaScript errors in Apdex calculations	The formula for computing the Apdex score is as follows: $[Satisfied\ requests + (Tolerating\ requests/2)]/Total\ samples$ By default, the count of <i>Satisfied</i> and <i>Tolerating requests</i> used for Apdex score computation includes the count of requests that experienced JavaScript errors. This is why, the Consider JavaScript errors in Apdex calculations flag is set to Yes by default. Since JavaScript errors also impact user experience, this default setting will result in an Apdex score that presents a near-real picture of the performance of a web site/web application. If you do not want the Apdex score to be tainted by error requests, then set this flag to <i>No</i>. In this case, requests with JavaScript errors will not be considered for Apdex calculations, even if such requests have a response time that is well-within (satisfied) or in violation (tolerating) of the Slow Transaction Cutoff.
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.eazycart.com/web/shopping/login.jsp, then http://www.eazycart.com will be the base URL or domain, <i>/web</i> will be the first URL segment, <i>/shopping</i> will be the second URL segment, and <i>/login.jsp</i> 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 <i>/web/shopping</i>. Here, <i>/web</i> corresponds to the specification 1 (URL segment 1) and <i>/shopping</i> 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.eazycart.com/web/shopping/products.jsp, http://www.eazycart.com/web/shopping/products/travel/bags.jsp, http://www.eazycart.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 <i>/web/shopping</i>. This way, the test will create different page groups based on each of the second-level URL segments in the managed web site – eg., <i>/web/movies</i>, <i>/web/travel</i>, <i>/web/reservations</i>, <i>/partner/contacts</i> etc. - and will report aggregated metrics for each group so created.

Parameter	Description
	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.eazycart.com/web/shopping/login.jsp, the test will report metrics for the descriptor <i>/shopping</i>. You can even set this parameter to 1,3. For a web site that contains URLs such as http://www.eazycart.com/web/shopping/products.jsp, http://www.eazycart.com/web/shopping/products/travel/bags.jsp, and http://www.eazycart.com/web/shopping/payment.jsp, this specification will result in the following descriptors: <i>/web/products</i>, <i>/web/products.jsp</i>, and <i>/web/payment.jsp</i>.
URL Patterns to be Ignored from Monitoring	By default, this test does not track requests to the following URL patterns: <i>*.js, *.css, *.jpeg, *.jpg, *.png</i>. 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 <i>.gif</i> and <i>.bmp</i> when monitoring, you need to alter the default specification as follows: <i>*.js, *.css, *.jpeg, *.jpg, *.png, *.gif, *.bmp</i> 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 <i>none</i> 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

Parameter	Description
	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: <i><Javascript error message>:<URL of the page/file where the message originated></i>. For instance, say that the <i>login.html</i> page in your web site runs a few Java scripts that throw <i>Object expected errors</i>, which you want the eG agent to ignore. In this case, your error specification can be as follows: <i>Object expected:http://www.eazycart.com/web/login.html</i>. Alternatively, you can provide only that text string with which the error message begins in your specification – eg., <i>Object:http://www.eazycart.com/web/login.html</i>. 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: <i>Object:login.html</i>. 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 <i>login.html</i> page points to a <i>validate.js</i> file, and you want to ignore the <i>Object expected errors</i> that this JS file throws, your error pattern specification will either be, <i>Object expected:validate.js</i>, or <i>Object:validate.js</i>. Multiple error message-URL combinations can also be provided as a comma-separated list. The format of your specification will be: <i><Javascript error message 1>:<Originating URL 1>,<Javascript error message 2>:<Originating URL 2>,...</i> For example, to ignore the <i>Object expected</i> and <i>Uncaught TypeError</i> errors in the <i>login.html</i> page, use the following specification: <i>Object:login.html,Uncaught:login.html</i> Likewise, to ignore the <i>Object expected</i> error in the <i>login.html</i> page and the <i>Uncaught TypeError</i> in the <i>validate.js</i> file, your specification will be: <i>Object:login.html,Uncaught:validate.js</i> If you want to ignore the <i>Uncaught TypeError</i> across all the pages of the web site, your specification will be as follows: <i>Uncaught TypeError:All</i> Note:

Parameter	Description
	When specifying the <i><Javascript error message></i> 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 <i>case-sensitive</i> pattern matching.
Include Page Type Categorization in Top-N DD	By default, the detailed diagnosis of this test reports the top-n transactions (in terms of <i>Avg page load time</i>) across page types. Accordingly, the Include Page Type Categorization in Top-N DD flag is set to No by default. If you want detailed diagnosis to report the top-n transactions per page type, then set this flag to Yes. In this case, if say the Maximum Healthy Transactions, Maximum Slow Transactions, and Maximum Error Transactions parameters are each set to 5, then the detailed diagnosis of this test will report the top-5 healthy, top-5 slow, and top-5 error transactions to Ajax pages, iFrames, Virtual pages, and base pages. As this may typically generate a large volume of data and increase the load on the eG database, you may want to set the 'Include Page Type Categorization in Top-N DD' to 'Yes', only if your eG database is sized sufficiently.
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, for each measurement period in the timeline chosen for detailed diagnosis. To identify the top 5, the eG agent sorts all healthy transactions in a measurement period 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 <i>all</i>. Before setting the value of this parameter to <i>all</i>, make sure that you have a well-sized eG database in place, as this setting will store details of every transaction that registers an <i>Avg page load time</i> 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	By default, this parameter is set to 5 indicating that the detailed diagnosis of this test

Parameter	Description
Transactions in DD	will display metrics related to the top-5 slow transactions, in terms of the <i>Average page load time</i>, for each measurement period in the timeline chosen for detailed diagnosis. To identify the top 5, the eG agent sorts all slow transactions in a measurement period in the descending order of their <i>Average page load time</i>, 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 <i>0</i>. To view all slow transactions, set the value of this parameter to <i>all</i>. Before setting the value of this parameter to <i>all</i>, make sure that you have a well-sized eG database in place, as this setting will store details of every transaction that registers an <i>Avg page load time</i> 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 <i>5</i>, indicating that the detailed diagnosis of this test will display metrics related to the top-5 transactions (per measurement period) that encountered JavaScript errors, based on when those errors occurred. To identify the top 5, the eG agent sorts all error transactions in a measurement period 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 <i>0</i>. To view all error transactions, set the value of this parameter to <i>all</i>. Before setting the value of this parameter to <i>all</i>, 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 <i>4000 millisecs</i> (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.

Parameter	Description
	Note: The default value of this parameter is the same as the default <i>Maximum threshold</i> setting of the <i>Avg page load time</i> measure – i.e., both are set to <i>4000 millisecs</i> 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 <i>Avg page load time</i>. 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 Transaction Cutoff is changed to <i>6000 millisecs</i>, change the <i>Maximum Threshold</i> of the <i>Avg page load time</i> measure to <i>6000 millisecs</i> as well.
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, Virtual Page, and/or iFrame views as well in the dashboard, by selecting the relevant options from this list box.
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 <i>0</i> - 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 <i>0</i>. These zero values will clearly indicate to users that there is no traffic to the monitored web site/web application.
DD Frequency	Refers to the frequency with which detailed diagnosis measures are to be generated for this test. The default is <i>3:1</i> . This indicates that, by default, detailed measures will be generated every third 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 <i>none</i> 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:

Parameter	Description
	• 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 genuine increase in traffic to the web site/web application. If the eG agent monitoring the <i>Real User Monitor</i> 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 <i>Page views</i> measure may indicate a sudden

Measurement	Measurement	Measurement Unit	Interpretation
			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 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.

Measurement	Measurement	Measurement Unit	Interpretation
			Ideally therefore, the value of this measure should be <i>1.0</i> . A value less than <i>1.0</i> 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. You may want to compare the values of the of the <i>Average browser initial request time</i>, <i>Average network time</i>, <i>Average server time</i>, <i>Average content download time</i>, and <i>Average browser render time</i> measures for that page group to know what is exactly causing pages of in the group to load slowly. If the <i>Average browser initial request time</i> is the highest, it can be attributed to a slowdown in redirection, AppCaching, or because the request spent too much time waiting on the browser. If the <i>Average network time</i> is the highest, it denotes that the network is the problem source. This in turn can be caused by TCP connection latencies, delays in domain look up, and slowdown in SSL handshake. If the <i>Average content download time</i> measure reports an abnormally high value,

Measurement	Measurement	Measurement Unit	Interpretation
			then it could be owing to slowness in downloading the HTML document or in building the DOM. On the other hand, if the <i>Average server time</i> 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. To know which pages in the group are slow, use the detailed diagnosis of this measure.
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 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 in this group has not been up to the mark. A value less than 50% is indicative of a serious problem, where most of the page view requests to 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

Measurement	Measurement	Measurement Unit	Interpretation
			<i>Slow page view percentage</i> with that of the <i>Javascript error page view percentage</i> for that page group. 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 <i>Slow page views</i> measure to know which pages in the group are slow and where these pages are losing time. If JavaScript errors are the problem, use the detailed diagnosis of the <i>JavaScript error view percentage</i> measure to know what errors occurred in which pages in the group.
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 being slow. Use the detailed diagnosis of the <i>Slow page views</i> 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

Measurement	Measurement	Measurement Unit	Interpretation
			indicates that users frequently experienced slowness when accessing pages in this group. To know which page views in this group 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 <i>Page views</i> 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 <i>Tolerating page views</i> and the <i>Frustrated page views</i>, 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 <i>Tolerating page views</i> and <i>Frustrated page views</i> measures to know which pages are slow and why.

Measurement	Measurement	Measurement Unit	Interpretation
Tolerating page views	Indicates the number of tolerating page views in this group.	Number	If the <i>Average page load time</i> of a page exceeds the Slow Transaction Cutoff configuration of this test, but is less than 4 times the Slow Transaction Cutoff (i.e., $< 4 * \text{Slow Transaction Cutoff}$), then such a page view is considered to be a <i>Tolerating page view</i>. Ideally, the value of this measure should be 0. A value higher than that of the <i>Satisfied page views</i> 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 <i>Average page load time</i> of a page is over 4 times the Slow Transaction Cutoff configuration of this test (i.e., $> 4 * \text{Slow Transaction Cutoff}$), 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 <i>Satisfied page views</i> measure is a cause for concern, as it implies that the experience of users to this page group 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

Measurement	Measurement	Measurement Unit	Interpretation
			backend?
Average browser initial request time	Indicates the interval between when a request was received by pages in this group, when it followed a redirect, to when it was processed in the AppCache.	ms	In Figure 3.205, the <i>Average browser initial request time</i> is the time spent between the <i>navigationStart</i> event to the <i>domainLookupStart</i> event. This also includes the time spent by the browser waiting for one event to end and the next to begin. In short, this measure is the sum of <i>Average redirect time</i>, <i>Average AppCache time</i>, and <i>Average browser wait time</i> measures. This means that if a request takes too long to follow a redirection, or if the AppCache takes too long to process the request, or if the request spends too much time on the browser for the previous request to complete, then the value of this measure will significantly increase. This in turn will impact user experience, and consequently, the <i>Apdex score</i>. This is why, if this measure reports an abnormal value for a page group, it is important that you compare the value of the <i>Average redirect time</i>, <i>Average AppCache time</i>, and <i>Average browser wait time</i> measures for the same page group, to figure out where the request spent maximum time - in redirection? in the AppCache? or when waiting on the browser?
Average server time	Indicates the interval between the start of processing of a request on the browser for this page group to when response is received.	ms	In Figure 3.205, the <i>Average server time</i> is the time spent between the <i>requestStart</i> event and <i>responseStart</i> event. Ideally, a low value is desired for this measure, as high values will certainly hurt the <i>Apdex score</i> of the web site/web application.

Measurement	Measurement	Measurement Unit	Interpretation
			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 <i>Average server time</i> will increase, leaving users with an unsatisfactory experience with the web site/web application. Note: This test uses the Navigation Timing API to measure web site performance. The Navigation Timing API typically exposes several properties that offer information about the time at which different page load events happen (see Figure 3.205) - eg., the <i>requestStart</i> event, the <i>responseStart</i> event, etc. This test uses the time stamps provided by the Navigation Timing API to compute and report the duration of page load events, so you can accurately identify where page loading is bottlenecked. The Navigation Timing API on Internet Explorer (IE) v11 reports an incorrect time stamp for the <i>requestStart</i> event of the page loading process (see Figure 3.205). As a result, for page view requests initiated from IE 11 browsers alone, eG RUM will report incorrect values for this measure.

Measurement	Measurement	Measurement Unit	Interpretation
			This issue was noticed in IE 11 in April 2019. It is recommended that you track hot fixes/patches released by Microsoft post April 2019, study the release notes of such fixes/patches, and determine if this bug has been fixed in any. If so, then you are advised to apply that fix/patch on IE 11 to resolve the issue. Until then, we recommend that you use the following workaround to accurately measure the <i>Average server time</i> of a page view request. ◦ Deploy eG BTM (Java/.NET, as the case may be) on the backend server hosting the target web site/web application. ◦ Use eG BTM to trace the path of transactions to the target web site/web application and enable the (Java or .NET) Business Transactions test to capture metrics on transaction performance. ◦ Next, use the detailed diagnostics reported by eG RUM to identify the page view requests coming from the IE 11 browser. Make a note of the values in the Request time, URL and Query Params columns of detailed diagnosis. ◦ Then, search the detailed diagnostics of the (Java or .NET) Business Transactions test for transactions with the same URL, Request time, and Query Params as reported by eG RUM.

Measurement	Measurement	Measurement Unit	Interpretation
			The response time that eG BTM reports for each of these transactions is the <i>Average server time</i> of those transactions. Note that this workaround applies only for those transaction URLs that are captured and reported as part of detailed diagnostics.
Average network 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 3.205, the time spent between <i>navigationStart</i> and <i>requestStart</i> makes up the <i>Average network time</i>. This includes the time to perform DNS lookups, the time to establish a TCP connection with the server, and the time to perform an SSL handshake. In other words, the value of this measure is nothing but the sum of the <i>Average DNS lookup time</i>, the <i>Average TCP connection time</i>, and the <i>Average SSL handshake time</i> 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 <i>Average DNS lookup time</i>, <i>Average TCP connection time</i>, and <i>Average SSL handshake time</i> measures to know to what this delay can be attributed – a delay in domain name resolution? Or a poor network connection to the server? or slowness in SSL negotiations between the browser and the server? Note:

Measurement	Measurement	Measurement Unit	Interpretation
			This test uses the Navigation Timing API to measure web site performance. The Navigation Timing API typically exposes several properties that offer information about the time at which different page load events happen (see Figure 3.205) - eg., the <code>requestStart</code> event, the <code>responseStart</code> event, etc. This test uses the time stamps provided by the Navigation Timing API to compute and report the duration of page load events, so you can accurately identify where page loading is bottlenecked. The Navigation Timing API on Internet Explorer (IE) v11 reports an incorrect time stamp for the <i>requestStart</i> event of the page loading process (see 3.6.5). As a result, for page view requests initiated from IE 11 browsers alone, eG RUM will report incorrect values for this measure. This issue was noticed in IE 11 in April 2019. It is recommended that you track hot fixes/patches released by Microsoft post April 2019, study the release notes of such fixes/patches, and determine if this bug has been fixed in any. If so, then you are advised to apply that fix/patch on IE 11 to resolve the issue.
Average content download time	Indicates the time to make the complete HTML document (DOM) available for JavaScript to apply rendering logic on the pages in this group.	ms	This is the time spent between the <code>responseStart</code> event (shown in 3.6.5) and the <code>documentContentLoadedEventStart</code> event (not shown in 3.6.5). The <code>documentContentLoadedEventStart</code> is typically fired just before the <code>domContentLoaded</code> event, which is just after browser has finished downloading and parsing all the scripts that had <code>defer</code> set and no <code>async</code> attribute.

Measurement	Measurement	Measurement Unit	Interpretation
			In summary, the <i>Average content download time</i> measure is the sum of the values of the <i>Average DOM download time</i> and <i>Average DOM processing time</i> measures. This means that an abnormal increase in any of the above-mentioned time values will increase the value of this measure. If content downloading takes unusually long for a request, then you must compare the values of the <i>Average DOM download time</i> and <i>Average DOM processing time</i> measures to figure out what is causing the delay - is it because of the poor responsiveness of server, cache, or local resource? or is it because DOM processing took too long?
Average browser render time	Indicates the time taken to complete the download of remaining resources, including images, and to finish rendering the pages in this group.	ms	A high value of this measure indicates that page rendering is taking too long. This can be attributed to an in-optimal HTML document architecture, complex CSS' (eg., deeply nested rules, slow selectors, complicated effects such as round borders, gradients etc.), and large sized images. If the <i>Average page load time</i> measure reports an abnormally high value, then you may want to compare the value of this measure with that of the <i>Average browser initial request time</i>, <i>Average network time</i>, <i>Average server time</i>, and <i>Average content download time</i> measures to nail the exact source of the bottleneck.
Average DOM download time	Indicates the time taken to download the	ms	The value of this measure is the time that elapsed between the responseStart and

Measurement	Measurement	Measurement Unit	Interpretation
	complete HTML document for this page group on the browser.		responseEnd events in 3.6.5. 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. Consequently, content download will be delayed, thus adversely impacting user experience with the web site/web application. Ideally therefore, the value of this measure should be low.
Average browser wait time	Indicates the time that requests to this page group spent on the browser, waiting for another request to complete.	ms	This is the sum of the time between every two consecutive events, starting with the navigationStart event till the requestStart event in 3.6.5. Typically, web browsers limit the number of active connections for each domain. Most modern browsers (eg., Chrome) support only six simultaneous requests/connections. In this case therefore, when the seventh request comes in, that request waits on the browser until the six requests sent previously are processed. The waiting time of the seventh request is the browser wait time. High browser wait time can prolong the browser's initial request time, thus adversely impacting the overall responsiveness of the web

Measurement	Measurement	Measurement Unit	Interpretation
			site/application. This is why, if the <i>Average browser initial request time</i> measure reports an abnormally high value, you will have to compare the values of the <i>Average browser wait time</i>, <i>Average redirection time</i>, and <i>Average AppCache time</i> measure to determine whether/not the initial request delay observed on the browser is because requests have been waiting on the browser for too long a time. Some of the means by which you can reduce browser waits are briefly discussed below: • Browsers such as Mozilla Firefox support upto 10 parallel requests. You may want to recommend such browsers for your web site/web application users, so that more requests are processed and fewer requests are queued on the browser, thus reducing browser wait time. • Web site/application developers can try domain sharding - i.e., split content across multiple domains. Typically, when a user connects to a web page, his or her browser scans the resulting HTML for resources to download. Normally these resources are supplied by a single domain - the domain providing the web page or a domain created specifically for resources. With domain sharding, the user's browser

Measurement	Measurement	Measurement Unit	Interpretation
			connects to two or more different domains to simultaneously download the resources needed to render the web page. This allows website/application to be delivered faster to users as they do not have to wait for the previous set of requests to end before beginning the next set.
Average redirection time	Indicates the time that requests to this page group spent in redirection before fetching the pages.	ms	In 3.6.5, this is the elapsed time between the redirectStart and redirectEnd events. URL redirection, also known as URL forwarding, is a technique to give a page, a form, or a whole web site/application, more than one URL address. Usually, web site administrators use URL redirection to: • Redirect users to the mobile version of the site • Redirect users to secured pages • Redirect users to the latest version of the resource/content • Redirect users to pages specific to their geo location • Redirecting canonical URLs Though redirects are useful, they have to be kept at a minimum, as each redirect on a page adds latency to the overall page load time. This is because, when a user enters a domain into the browser and hits enter, the DNS resolution process is triggered and the domain is resolved to its corresponding IP address in a few

Measurement	Measurement	Measurement Unit	Interpretation
			milliseconds. If the landing page has another redirect, then the browser repeats the entire DNS resolution process once again to guide the user to the correct web page. The multiple redirect requests are taxing on the browser resources and slow down the page load. Web page load time is also affected by internal redirects; for example, if the page tries to load content from a URL that has been redirected to newer or updated content, then the browser must create additional requests to fetch the valid content. These redirects result in additional round trips between the browser and the web server which pushes the load time higher; the perceived performance is degraded every time the browser encounters a redundant redirect. Web site/application performance is also impacted if redirects are not implemented correctly. Some of the common redirect errors are: • Multiple redirects: The higher the number of redirects on a page, the higher is its page load time. • Invalid redirects: There are often instances where the web site administrator sets up bulk redirects without verifying the validity of the redirects. The site may also have old redirects that were never cleaned up. This can cause several issues on the site like broken links and 404s.

Measurement	Measurement	Measurement Unit	Interpretation
			• Redirect loop: When there are several redirects on the page that are linked to each other, it creates a chain of redirects which may loop back to the same URL that initiated the redirect. This results in a redirect loop error and the user will not be able to access the site. Therefore, if you find that the value of the <i>Average page load time</i> measure is abnormally high owing to an unusually high value for the Average redirection time measure, then make sure you follow the best practices outlined below while implementing redirects, so you can significantly reduce page load time and improve user experience: • Avoid redundant redirects: It's recommended to avoid redirects where possible and to use this method only when absolutely needed. This will cut down unnecessary overhead and improve the perceived performance of the page. • Chain redirects: When a URL is linked to another URL, this creates a chained redirect. Each URL added to the chain adds latency to the page. Chained redirects have a negative impact not only on page speed, but also SEO. • Clean-up redirects: You may have hundreds of redirects on your website

Measurement	Measurement	Measurement Unit	Interpretation
			and it could be one of the main factors affecting page speed. Old redirects may conflict with new URLs, backlinks can cause odd errors on the page. It is recommended to verify all the redirects you have set up and to remove the ones that are no longer needed. Retain the old links that have major referral traffic, while those that are rarely accessed can be removed. This exercise will help improve page speed significantly.
Average AppCache time	Indicates the time taken to check whether/not the requests to this page group can be serviced by the AppCache.	ms	HTML5 provides an application caching mechanism that lets web-based applications run offline. Developers can use the Application Cache (AppCache) interface to specify resources that the browser should cache and make available to offline users. Applications that are cached load and work correctly even if users click the refresh button when they are offline. Using an application cache gives an application the following benefits: • Offline browsing: users can navigate a site even when they are offline. • Speed: cached resources are local, and therefore load faster. • Reduced server load: the browser only downloads resources that have changed from the server. To enable the application cache for an application, you must include the manifest attribute in the <html> element in your application's pages. The manifest attribute references a cache manifest file, which is

Measurement	Measurement	Measurement Unit	Interpretation
			a text file that lists resources (files) that the browser should cache for your application. The browser does not cache pages that do not contain the manifest attribute, unless such pages are explicitly listed in the manifest file itself. You do not need to list all the pages you want cached in the manifest file, the browser implicitly adds every page that the user visits and that has the manifest attribute set to the application cache. When the browser visits a document that includes the manifest attribute, if no application cache exists, the browser loads the document and then fetches all the entries listed in the manifest file, creating the first version of the application cache. Subsequent visits to that document cause the browser to load the document and other assets specified in the manifest file from the application cache. If the manifest file has changed, all the files listed in the manifest—as well as those already added to the cache—are fetched into a temporary cache. Once all the files have been successfully retrieved, they are moved into the real offline cache automatically. Since the document has already been loaded into the browser from the cache, the updated document will not be rendered until the document is reloaded (either manually or programmatically). A high value of this measure signifies that requests are spending too much time in the AppCache. This also introduces page

Measurement	Measurement	Measurement Unit	Interpretation
			loading latencies, which have an adverse effect on user-perceived performance of a web site/application. Common reasons for AppCaching issues and their practical solutions are detailed below: • If the media type is not set, then AppCache will not work. To avoid this, make sure that the manifest file is always served under the correct media type of text/cache-manifest. • If the manifest file is not served to the web browser from the same origin as the host page, the manifest file will fail to load. To avoid this, make sure that the manifest file is always served from the same origin as the host page. However, note that the manifest file can hold reference to resources to be cached from other domains. • The relative URLs that you mention in the manifest are relative to the manifest file and not to the document where you reference the manifest file. If you make this error when the manifest and the reference are not in the same path, the resources will fail to load and in turn the manifest file will not be loaded. This will stall appcaching. • Any change made to the manifest file will cause the entire set of files to be downloaded again. Moreover, if a

Measurement	Measurement	Measurement Unit	Interpretation
			manifest file is added to a HTML file, it forces all resources to be downloaded synchronously as soon as the manifest file is downloaded. As a result, resources that may not yet be required, such as JavaScript or an image below the fold, will be downloaded at the start of the page. This can increase page load time significantly. The solution to this is to load the Application Cache from a simple HTML file loaded in an iframe. This not only avoids caching dynamic HTML but also avoids the Application Cache being downloaded asynchronously after the page load has completed.
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 <i>Average network time</i> and <i>Average page load time</i> 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 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 <i>Average network time</i> too will increase, thus impacting page load time and overall user experience with the web site/web application.
Average SSL handshake time	Indicates the time taken by requests to this page group to complete the SSL handshake.	ms	An SSL handshake happens when a browser makes a secure request for content, also known as an encrypted HTTPS connection. The user's browser and server negotiate encrypted keys and certificates to establish a secure connection between each other. Because this SSL negotiation requires exchanges between the browser and your server, it

Measurement	Measurement	Measurement Unit	Interpretation
			increases the time spent by the request on the network (i.e., it adds to the value of the Average network time measure). This in turn increases page load time. In fact, an SSL handshake, along with DNS lookup and TCP handshake add three round trips to the page load time. A quick fix to this is to use HTTP/2. HTTP/2 can use caching to reduce SSL setup to only one round trip.

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 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

Configurable parameters for the test

Parameter	Description
Test Period	How often should the test be executed.
Proxy Host	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 <i>none</i> , indicating that the eG agent does not communicate with the collector via a proxy.
Proxy Port	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 <i>none</i> , indicating that the eG agent does not communicate with the collector via a proxy.
Proxy Username and Proxy Password	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 Proxy Username and Proxy Password . If no proxy server is used or if the proxy server used does not require authentication, then set the ProxyUsername and ProxyPassword to <i>none</i> .
Confirm Password	Confirm the Proxy Password by retyping it here. Note: If you Reconfigure the test later to change the values of the Proxy Username , Proxy Password , 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

Parameter	Description
	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. MAXIMUM ALLOWED PAGE VISITS PER DAY 100000 Configuring the number of allowed page visits By default, the Maximum allowed page visits per day is set to <i>100000</i>. This implies that the test will consider only the first <i>100000</i> requests in a day for monitoring. All page visits beyond <i>100000</i> will by default be excluded from the test's monitoring purview. You can increase or decrease this limit, if you so need.
Consider JavaScript errors in Apdex calculations	The formula for computing the Apdex score is as follows: $[Satisfied\ requests + (Tolerating\ requests/2)]/Total\ samples$ By default, the count of <i>Satisfied</i> and <i>Tolerating requests</i> used for Apdex score computation includes the count of requests that experienced JavaScript errors. This is why, the Consider JavaScript errors in Apdex calculations flag is set to Yes by default. Since JavaScript errors also impact user experience, this default setting will result in an Apdex score that presents a near-real picture of the performance of a web site/web application. If you do not want the Apdex score to be tainted by error requests, then set this flag to <i>No</i>. In this case, requests with JavaScript errors will not be considered for Apdex calculations, even if such requests have a response time that is well-within (satisfied) or in violation (tolerating) of the Slow Transaction Cutoff.
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.eazycart.com/web/shopping/login.jsp, then http://www.eazycart.com will be the base URL or domain, <i>/web</i> will be the first URL segment, <i>/shopping</i> will be the second URL segment, and <i>/login.jsp</i> 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

Parameter	Description
	time metrics to all instrumented web pages (i.e., web pages with the code snippet) under the URL <i>/web/shopping</i>. Here, <i>/web</i> corresponds to the specification 1 (URL segment 1) and <i>/shopping</i> 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.eazycart.com/web/shopping/products.jsp, http://www.eazycart.com/web/shopping/products/travel/bags.jsp, http://www.eazycart.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 <i>/web/shopping</i>. This way, the test will create different page groups based on each of the second-level URL segments in the managed web site – eg., <i>/web/movies</i>, <i>/web/travel</i>, <i>/web/reservations</i>, <i>/partner/contacts</i> 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.eazycart.com/web/shopping/login.jsp, the test will report metrics for the descriptor <i>/shopping</i>. You can even set this parameter to 1,3. For a web site that contains URLs such as http://www.eazycart.com/web/shopping/products.jsp, http://www.eazycart.com/web/shopping/products/travel/bags.jsp, and http://www.eazycart.com/web/shopping/payment.jsp, this specification will result in the following descriptors: <i>/web/products</i>, <i>/web/products.jsp</i>, and <i>/web/payment.jsp</i>.
URL Patterns to be Ignored from Monitoring	By default, this test does not track requests to the following URL patterns: <i>*.js, *.css, *.jpeg, *.jpg, *.png</i>. 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 <i>.gif</i> and <i>.bmp</i> when monitoring, you need to alter the default specification as follows: <i>*.js, *.css, *.jpeg, *.jpg, *.png, *.gif, *.bmp</i> 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

Parameter	Description
	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 <i>none</i> 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: <i><Javascript error message>:<URL of the page/file where the message originated></i>. For instance, say that the <i>login.html</i> page in your web site runs a few Java scripts that throw <i>Object expected errors</i>, which you want the eG agent to ignore. In this case, your error specification can be as follows: <i>Object expected:http://www.eazycart.com/web/login.html</i>. Alternatively, you can provide only that text string with which the error message begins in your specification – eg., <i>Object:http://www.eazycart.com/web/login.html</i>. 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: <i>Object:login.html</i>. 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 <i>login.html</i> page points to a <i>validate.js</i> file, and you want to ignore the Object expected errors that this JS file throws, your error pattern specification will either be, <i>Object expected:validate.js</i>, or <i>Object:validate.js</i>. Multiple error message-URL combinations can also be provided as a comma-separated list. The format of your specification will be: <i><Javascript error message 1>:<Originating URL 1>,<Javascript error message 2>:<Originating URL 2>,...</i>

Parameter	Description
	For example, to ignore the <i>Object expected</i> and <i>Uncaught TypeError</i> errors in the <i>login.html</i> page, use the following specification: <i>Object:login.html,Uncaught:login.html</i> Likewise, to ignore the <i>Object expected</i> error in the <i>login.html</i> page and the <i>Uncaught TypeError</i> in the <i>validate.js</i> file, your specification will be: <i>Object:login.html,Uncaught:validate.js</i> If you want to ignore the <i>Uncaught TypeError</i> across all the pages of the web site, your specification will be as follows: <i>Uncaught TypeError:All</i> Note: When specifying the <i><Javascript error message></i> 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 <i>case-sensitive</i> pattern matching.
Include Page Type Categorization in Top-N DD	By default, the detailed diagnosis of this test reports the top-n transactions (in terms of <i>Avg page load time</i>) across page types. Accordingly, the Include Page Type Categorization in Top-N DD flag is set to No by default. If you want detailed diagnosis to report the top-n transactions per page type, then set this flag to Yes. In this case, if say the Maximum Healthy Transactions, Maximum Slow Transactions, and Maximum Error Transactions parameters are each set to 5, then the detailed diagnosis of this test will report the top-5 healthy, top-5 slow, and top-5 error transactions to Ajax pages, iFrames, Virtual pages, and base pages. As this may typically generate a large volume of data and increase the load on the eG database, you may want to set the 'Include Page Type Categorization in Top-N DD' to 'Yes', only if your eG database is sized sufficiently.
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, for each measurement period in the timeline chosen for detailed diagnosis. To identify the top 5, the eG agent sorts all healthy transactions in a measurement period 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

Parameter	Description
	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 <i>0</i>. To view all healthy transactions, set the value of this parameter to <i>all</i>. Before setting the value of this parameter to <i>all</i>, make sure that you have a well-sized eG database in place, as this setting will store details of every transaction that registers an <i>Avg page load time</i> 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 <i>5</i> indicating that the detailed diagnosis of this test will display metrics related to the top-5 slow transactions, in terms of the <i>Average page load time</i>, for each measurement period in the timeline chosen for detailed diagnosis. To identify the top 5, the eG agent sorts all slow transactions in a measurement period in the descending order of their <i>Average page load time</i>, 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 <i>0</i>. To view all slow transactions, set the value of this parameter to <i>all</i>. Before setting the value of this parameter to <i>all</i>, make sure that you have a well-sized eG database in place, as this setting will store details of every transaction that registers an <i>Avg page load time</i> 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 <i>5</i>, indicating that the detailed diagnosis of this test will display metrics related to the top-5 transactions (per measurement period) that encountered JavaScript errors, based on when those errors occurred. To identify the top 5, the eG agent sorts all error transactions in a measurement period 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-

Parameter	Description
	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 <i>0</i>. To view all error transactions, set the value of this parameter to <i>all</i>. Before setting the value of this parameter to <i>all</i>, 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 <i>4000 millisecs</i> (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 <i>Maximum threshold</i> setting of the <i>Avg page load time</i> measure – i.e., both are set to <i>4000 millisecs</i> 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 <i>Avg page load time</i>. 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 Transaction Cutoff is changed to <i>6000 millisecs</i>, change the <i>Maximum Threshold</i> of the <i>Avg page load time</i> measure to <i>6000 millisecs</i> as well.
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, Virtual Page, and/or iFrame views as well in the dashboard, by selecting the relevant options from this list box.
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 <i>0</i> - 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 <i>0</i>. These zero values will clearly indicate to users that there is no traffic to the monitored web site/web application.

Parameter	Description
DD Frequency	Refers to the frequency with which detailed diagnosis measures are to be generated for this test. The default is 3:1. This indicates that, by default, detailed measures will be generated every third 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 <i>none</i> 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 <i>Real User</i>

Measurement	Description	Measurement Unit	Interpretation
			<i>Monitor</i> 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 <i>Page views</i> 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.

Measurement	Description	Measurement Unit	Interpretation
			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 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 could be the region that is causing the slowness. You may want to compare the values of the of the <i>Average browser initial request time</i>, <i>Average network time</i>, <i>Average server time</i>, <i>Average content download time</i>, and <i>Average browser</i>

Measurement	Description	Measurement Unit	Interpretation
			<i>render time</i> measures for that region to know what is exactly causing users from that region to experience slowness. If the <i>Average browser initial request time</i> is the highest, it can be attributed to a slowdown in redirection, AppCaching, or because the request spent too much time waiting on the browser. If the <i>Average network time</i> is the highest, it denotes that the network is the problem source. This in turn can be caused by TCP connection latencies, delays in domain look up, and slowdown in SSL handshake. If the <i>Average content download time</i> measure reports an abnormally high value, then it could be owing to slowness in downloading the HTML document or in building the DOM. On the other hand, if the <i>Average server time</i> 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. To know which pages are slow, use the detailed diagnosis of this measure.
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	Indicates the percentage	Percent	The value of this measure indicates the

Measurement	Description	Measurement Unit	Interpretation
percentage	of page views that delivered a satisfactory experience to users from this region.		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 experience of users from this region has not been up to the mark. A value less than 50% is indicative of a serious problem, where most of the page view requests from this region are either slow or have encountered Javascript errors. Under such circumstances, to know what exactly is affecting the experience of users from this region, compare the value of the <i>Slow page view percentage</i> with that of the <i>Javascript error page view percentage</i> for that region. 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 <i>Slow page views</i> measure to know which pages accessed by users from this region are slow and where these pages are losing time. If JavaScript errors are the problem, use the detailed diagnosis of the <i>JavaScript error view percentage</i> measure to know what errors occurred in which pages accessed from this region.
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 <i>Slow page views</i> measure to identify the slow pages and

Measurement	Description	Measurement Unit	Interpretation
			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 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. To know which pages were slow when accessed from this region 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

Measurement	Description	Measurement Unit	Interpretation
			diagnosis of this measure. Ideally, the value of this measure should be the same as that of the <i>Page views</i> 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 <i>Tolerating page views</i> and the <i>Frustrated page views</i>, it is a clear indicator that the experience of users from this region has been below-par. In such a case, use the detailed diagnosis of the <i>Tolerating page views</i> and <i>Frustrated page views</i> 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 <i>Average page load time</i> of a page exceeds the Slow Transaction Cutoff configuration of this test, but is less than 4 times the Slow Transaction Cutoff (i.e., $< 4 * \text{Slow Transaction Cutoff}$), then such a page view is considered to be a <i>Tolerating page view</i>. Ideally, the value of this measure should be 0. A value higher than that of the <i>Satisfied page views</i> measure is a cause for concern, as it implies that the overall experience of users from 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	Number	If the <i>Average page load time</i> of a page is over 4 times the Slow Transaction

Measurement	Description	Measurement Unit	Interpretation
	experienced by users in this region.		Cutoff 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 <i>Satisfied page views</i> measure is a cause for concern, as it implies that the experience of users from this region 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 browser initial request time	Indicates the interval between when a request was received from users in this region, when it followed a redirect, to when it was processed in the AppCache.	ms	In Figure 3.205, the <i>Average browser initial request time</i> is the time spent between the navigationStart event to the domainLookupStart event. This also includes the time spent by the browser waiting for one event to end and the next to begin. In short, this measure is the sum of <i>Average redirect time</i>, <i>Average AppCache time</i>, and <i>Average browser wait time</i> measures. This means that if a request takes too long to follow a redirection, or if the AppCache takes too long to process the request, or if the request spends too much time on the browser for the previous request to complete, then the value of this measure will significantly increase. This in turn will impact user experience, and consequently, the <i>Apdex score</i>. This is why, if this measure reports an abnormal value for a region, it is important

Measurement	Description	Measurement Unit	Interpretation
			that you compare the value of the <i>Average redirect time</i> , <i>Average AppCache time</i> , and <i>Average browser wait time</i> measures for the same region, to figure out where the request spent maximum time - in redirection? in the AppCache? or when waiting on the browser?
Average server time	Indicates the interval between the start of processing of a request from this region on the browser to when response is received.	ms	In Figure 3.205, the <i>Average server time</i> is the time spent between the <i>requestStart</i> event and <i>responseStart</i> event. Ideally, a low value is desired for this measure, as high values will certainly hurt the <i>Apdex score</i> 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 <i>Average server time</i> will increase, leaving users with an unsatisfactory experience with the web site/web application. Note: This test uses the Navigation Timing API to measure web site performance. The Navigation Timing API typically exposes several properties that offer information about the time at which

Measurement	Description	Measurement Unit	Interpretation
			different page load events happen (see Figure 3.205) - eg., the <i>requestStart</i> event, the <i>responseStart</i> event, etc. This test uses the time stamps provided by the Navigation Timing API to compute and report the duration of page load events, so you can accurately identify where page loading is bottlenecked. The Navigation Timing API on Internet Explorer (IE) v11 reports an incorrect time stamp for the <i>requestStart</i> event of the page loading process (see Figure 3.205). As a result, for page view requests initiated from IE 11 browsers alone, eG RUM will report incorrect values for this measure. This issue was noticed in IE 11 in April 2019. It is recommended that you track hot fixes/patches released by Microsoft post April 2019, study the release notes of such fixes/patches, and determine if this bug has been fixed in any. If so, then you are advised to apply that fix/patch on IE 11 to resolve the issue. Until then, we recommend that you use the following workaround to accurately measure the <i>Average server time</i> of a page view request. - Deploy eG BTM (Java/.NET, as the case may be) on the backend server hosting the target web site/web application. - Use eG BTM to trace the path of transactions to the target web site/web application and enable the (Java or .NET) Business

Measurement	Description	Measurement Unit	Interpretation
			Transactions test to capture metrics on transaction performance. - Next, use the detailed diagnostics reported by eG RUM to identify the page view requests coming from the IE 11 browser. Make a note of the values in the Request time, URL and Query Params columns of detailed diagnosis. - Then, search the detailed diagnostics of the (Java or .NET) Business Transactions test for transactions with the same URL, Request time, and Query Params as reported by eG RUM. - The response time that eG BTM reports for each of these transactions is the <i>Average server time</i> of those transactions. Note that this workaround applies only for those transaction URLs that are captured and reported as part of detailed diagnostics.
Average network time	Indicates the elapsed time since a user from this region initiates a request to the web site/application and the start of fetching the response document from it.	ms	In Figure 3.205, the time spent between <i>navigationStart</i> and <i>requestStart</i> makes up the <i>Average network time</i>. This includes the time to perform DNS lookups, the time to establish a TCP connection with the server, and the time to perform an SSL handshake. In other words, the value of this measure is nothing but the sum of the <i>Average DNS lookup time</i>, the <i>Average TCP connection time</i>, and the <i>Average SSL handshake time</i> measures.

Measurement	Description	Measurement Unit	Interpretation
			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 <i>Average DNS lookup time</i> , <i>Average TCP connection time</i>, and <i>Average SSL handshake time</i> measures to know to what this delay can be attributed – a delay in domain name resolution? Or a poor network connection to the server? or slowness in SSL negotiations between the browser and the server? Note: This test uses the Navigation Timing API to measure web site performance. The Navigation Timing API typically exposes several properties that offer information about the time at which different page load events happen (see Figure 3.205) - eg., the <i>requestStart</i> event, the <i>responseStart</i> event, etc. This test uses the time stamps provided by the Navigation Timing API to compute and report the duration of page load events, so you can accurately identify where page loading is bottlenecked. The Navigation Timing API on Internet Explorer (IE) v11 reports an incorrect time stamp for the <i>requestStart</i> event of the page loading process (see 3.6.6). As a result, for page view requests initiated from IE 11 browsers alone, eG RUM will report incorrect values for this measure. This issue was noticed in IE 11 in April 2019. It is recommended that you track

Measurement	Description	Measurement Unit	Interpretation
			hot fixes/patches released by Microsoft post April 2019, study the release notes of such fixes/patches, and determine if this bug has been fixed in any. If so, then you are advised to apply that fix/patch on IE 11 to resolve the issue.
Average content download time	Indicates the time to make the complete HTML document (DOM) available for JavaScript to apply rendering logic on the pages accessed from this region.	ms	This is the time spent between the responseStart event (shown in 3.6.6) and the documentContentLoadedEventStart event (not shown in 3.6.6). The documentContentLoadedEventStart is typically fired just before the domContentLoaded event, which is just after browser has finished downloading and parsing all the scripts that had defer set and no async attribute. In summary, the <i>Average content download time</i> measure is the sum of the values of the <i>Average DOM download time</i> and <i>Average DOM processing time</i> measures. This means that an abnormal increase in any of the above-mentioned time values will increase the value of this measure. If content downloading takes unusually long for a request, then you must compare the values of the <i>Average DOM download time</i> and <i>Average DOM processing time</i> measures to figure out what is causing the delay - is it because of the poor responsiveness of server, cache, or local resource? or is it because DOM processing took too long?
Average browser render time	Indicates the time taken to complete the download of remaining	ms	A high value of this measure indicates that page rendering is taking too long. This can be attributed to an in-optimal

Measurement	Description	Measurement Unit	Interpretation
	resources, including images, and to finish rendering the pages accessed from this region.		HTML document architecture, complex CSS' (eg., deeply nested rules, slow selectors, complicated effects such as round borders, gradients etc.), and large sized images. If the <i>Average page load time</i> measure reports an abnormally high value, then you may want to compare the value of this measure with that of the <i>Average browser initial request time</i>, <i>Average network time</i>, <i>Average server time</i>, and <i>Average content download time</i> measures to nail the exact source of the bottleneck.
Average DOM download time	Indicates the time taken to download the complete HTML document for page view requests from this region, on the browser.	ms	The value of this measure is the time that elapsed between the responseStart and responseEnd events in 3.6.6. 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 accessed from 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. Consequently, content download will be delayed, thus adversely impacting user experience with the web site/web application. Ideally therefore, the value of this measure should be low.
Average browser wait time	Indicates the time that requests from this region spent on the browser, waiting for another	ms	This is the sum of the time between every two consecutive events, starting with the navigationStart event till the requestStart event in 3.6.6.

Measurement	Description	Measurement Unit	Interpretation
	request to complete.		Typically, web browsers limit the number of active connections for each domain. Most modern browsers (eg., Chrome) support only six simultaneous requests/connections. In this case therefore, when the seventh request comes in, that request waits on the browser until the six requests sent previously are processed. The waiting time of the seventh request is the browser wait time. High browser wait time can prolong the browser's initial request time, thus adversely impacting the overall responsiveness of the web site/application. This is why, if the <i>Average browser initial request time</i> measure reports an abnormally high value, you will have to compare the values of the <i>Average browser wait time</i>, <i>Average redirection time</i>, and <i>Average AppCache time</i> measure to determine whether/not the initial request delay observed on the browser is because requests have been waiting on the browser for too long a time. Some of the means by which you can reduce browser waits are briefly discussed below: • Browsers such as Mozilla Firefox support upto 10 parallel requests. You may want to recommend such browsers for your web site/web application users, so that more requests are processed and fewer requests are queued on the browser,

Measurement	Description	Measurement Unit	Interpretation
			thus reducing browser wait time. Web site/application developers can try domain sharding - i.e., split content across multiple domains. Typically, when a user connects to a web page, his or her browser scans the resulting HTML for resources to download. Normally these resources are supplied by a single domain - the domain providing the web page or a domain created specifically for resources. With domain sharding, the user's browser connects to two or more different domains to simultaneously download the resources needed to render the web page. This allows website/application to be delivered faster to users as they do not have to wait for the previous set of requests to end before beginning the next set.
Average redirection time	Indicates the time that requests from this region spent in redirection before fetching the pages.	ms	In 3.6.6, this is the elapsed time between the redirectStart and redirectEnd events. URL redirection, also known as URL forwarding, is a technique to give a page, a form, or a whole web site/application, more than one URL address. Usually, web site administrators use URL redirection to: - Redirect users to the mobile version of the site - Redirect users to secured pages

Measurement	Description	Measurement Unit	Interpretation
			• Redirect users to the latest version of the resource/content • Redirect users to pages specific to their geo location • Redirecting canonical URLs Though redirects are useful, they have to be kept at a minimum, as each redirect on a page adds latency to the overall page load time. This is because, when a user enters a domain into the browser and hits enter, the DNS resolution process is triggered and the domain is resolved to its corresponding IP address in a few milliseconds. If the landing page has another redirect, then the browser repeats the entire DNS resolution process once again to guide the user to the correct web page. The multiple redirect requests are taxing on the browser resources and slow down the page load. Web page load time is also affected by internal redirects; for example, if the page tries to load content from a URL that has been redirected to newer or updated content, then the browser must create additional requests to fetch the valid content. These redirects result in additional round trips between the browser and the web server which pushes the load time higher; the perceived performance is degraded every time the browser encounters a redundant redirect. Web site/application performance is also impacted if redirects are not implemented correctly. Some of the common redirect errors are:

Measurement	Description	Measurement Unit	Interpretation
			• Multiple redirects: The higher the number of redirects on a page, the higher is its page load time. • Invalid redirects: There are often instances where the web site administrator sets up bulk redirects without verifying the validity of the redirects. The site may also have old redirects that were never cleaned up. This can cause several issues on the site like broken links and 404s. • Redirect loop: When there are several redirects on the page that are linked to each other, it creates a chain of redirects which may loop back to the same URL that initiated the redirect. This results in a redirect loop error and the user will not be able to access the site. Therefore, if you find that the value of the <i>Average page load time</i> measure is abnormally high owing to an unusually high value for the <i>Average redirection time</i> measure, then make sure you follow the best practices outlined below while implementing redirects, so you can significantly reduce page load time and improve user experience: • Avoid redundant redirects: It's recommended to avoid redirects where possible and to use this method only

Measurement	Description	Measurement Unit	Interpretation
			when absolutely needed. This will cut down unnecessary overhead and improve the perceived performance of the page. - Chain redirects: When a URL is linked to another URL, this creates a chained redirect. Each URL added to the chain adds latency to the page. Chained redirects have a negative impact not only on page speed, but also SEO. - Clean-up redirects: You may have hundreds of redirects on your website and it could be one of the main factors affecting page speed. Old redirects may conflict with new URLs, backlinks can cause odd errors on the page. It is recommended to verify all the redirects you have set up and to remove the ones that are no longer needed. Retain the old links that have major referral traffic, while those that are rarely accessed can be removed. This exercise will help improve page speed significantly.
Average AppCache time	Indicates the time taken to check whether/not the requests from this region can be serviced by the AppCache.	ms	HTML5 provides an application caching mechanism that lets web-based applications run offline. Developers can use the Application Cache (AppCache) interface to specify resources that the browser should cache and make available to offline users. Applications that are cached load and work correctly even if users click the refresh button when they are offline. Using an application cache gives an application the following benefits:

Measurement	Description	Measurement Unit	Interpretation
			• Offline browsing: users can navigate a site even when they are offline. • Speed: cached resources are local, and therefore load faster. • Reduced server load: the browser only downloads resources that have changed from the server. To enable the application cache for an application, you must include the manifest attribute in the <html> element in your application's pages. The manifest attribute references a cache manifest file, which is a text file that lists resources (files) that the browser should cache for your application. The browser does not cache pages that do not contain the manifest attribute, unless such pages are explicitly listed in the manifest file itself. You do not need to list all the pages you want cached in the manifest file; the browser implicitly adds every page that the user visits and that has the manifest attribute set to the application cache. When the browser visits a document that includes the manifest attribute, if no application cache exists, the browser loads the document and then fetches all the entries listed in the manifest file, creating the first version of the application cache. Subsequent visits to that document cause the browser to load the document and other assets specified in the manifest file from the application cache. If the manifest file has changed, all the

Measurement	Description	Measurement Unit	Interpretation
			files listed in the manifest—as well as those already added to the cache—are fetched into a temporary cache. Once all the files have been successfully retrieved, they are moved into the real offline cache automatically. Since the document has already been loaded into the browser from the cache, the updated document will not be rendered until the document is reloaded (either manually or programmatically). A high value of this measure signifies that requests are spending too much time in the AppCache. This also introduces page loading latencies, which have an adverse effect on user-perceived performance of a web site/application. Common reasons for AppCaching issues and their practical solutions are detailed below: • If the media type is not set, then AppCache will not work. To avoid this, make sure that the manifest file is always served under the correct media type of text/cache-manifest. • If the manifest file is not served to the web browser from the same origin as the host page, the manifest file will fail to load. To avoid this, make sure that the manifest file is always served from the same origin as the host page. However, note that the manifest file can hold reference to resources to be cached from other domains.

Measurement	Description	Measurement Unit	Interpretation
			- The relative URLs that you mention in the manifest are relative to the manifest file and not to the document where you reference the manifest file. If you make this error when the manifest and the reference are not in the same path, the resources will fail to load and in turn the manifest file will not be loaded. This will stall appcaching. - Any change made to the manifest file will cause the entire set of files to be downloaded again. Moreover, if a manifest file is added to a HTML file, it forces all resources to be downloaded synchronously as soon as the manifest file is downloaded. As a result, resources that may not yet be required, such as JavaScript or an image below the fold, will be downloaded at the start of the page. This can increase page load time significantly. The solution to this is to load the Application Cache from a simple HTML file loaded in an iframe. This not only avoids caching dynamic HTML but also avoids the Application Cache being downloaded asynchronously after the page load has completed.
Average DNS lookup time	Indicates the time taken by the browser to perform the domain lookup for connecting to the pages requested by this region.	ms	A high value for this measure will not only affect DNS lookup, but will also impact the <i>Average network time</i> and <i>Average page load time</i> 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	ms	A bad network connection between the browser client and the server can delay

Measurement	Description	Measurement Unit	Interpretation
	establish a TCP connection with the server for accessing pages from this region.		TCP connections to the server. As a result, the <i>Average network time</i> too will increase, thus impacting page load time and overall user experience with the web site/web application.
Average SSL handshake time	Indicates the time taken by requests from this region to complete the SSL handshake.	ms	An SSL handshake happens when a browser makes a secure request for content, also known as an encrypted HTTPS connection. The user's browser and server negotiate encrypted keys and certificates to establish a secure connection between each other. Because this SSL negotiation requires exchanges between the browser and your server, it increases the time spent by the request on the network (i.e., it adds to the value of the Average network time measure). This in turn increases page load time. In fact, an SSL handshake, along with DNS lookup and TCP handshake add three round trips to the page load time. A quick fix to this is to use HTTP/2. HTTP/2 can use caching to reduce SSL setup to only one round trip.

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

Configurable parameters for the test

Parameter	Description
Test Period	How often should the test be executed.
Proxy Host	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 <i>none</i> , indicating that the eG agent does not communicate with the collector via a proxy.
Proxy Port	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 <i>none</i> , indicating that the eG agent does not communicate with the collector via a proxy.
Proxy Username and Proxy Password	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 Proxy Username and Proxy Password . If no proxy server is used or if the proxy server used does not require authentication, then set the ProxyUsername and ProxyPassword to <i>none</i> .
Confirm Password	Confirm the Proxy Password by retyping it here.
Note: If you Reconfigure the test later to change the values of the Proxy Username , Proxy Password , Confirm Password parameters, then such changes will be effected only if the eG remote agent monitoring the Real User Monitor component is restarted.	

Parameter	Description
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.  Configuring the number of allowed page visits By default, the Maximum allowed page visits per day is set to <i>100000</i>. This implies that the test will consider only the first <i>100000</i> requests in a day for monitoring. All page visits beyond <i>100000</i> will by default be excluded from the test's monitoring purview. You can increase or decrease this limit, if you so need.
Consider JavaScript errors in Apdex calculations	The formula for computing the Apdex score is as follows: $[Satisfied\ requests + (Tolerating\ requests/2)]/Total\ samples$ By default, the count of <i>Satisfied</i> and <i>Tolerating requests</i> used for Apdex score computation includes the count of requests that experienced JavaScript errors. This is why, the Consider JavaScript errors in Apdex calculations flag is set to Yes by default. Since JavaScript errors also impact user experience, this default setting will result in an Apdex score that presents a near-real picture of the performance of a web site/web application. If you do not want the Apdex score to be tainted by error requests, then set this flag to <i>No</i>. In this case, requests with JavaScript errors will not be considered for Apdex calculations, even if such requests have a response time that is well-within (satisfied) or in violation (tolerating) of the Slow Transaction Cutoff.
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.eazycart.com/web/shopping/login.jsp, then http://www.eazycart.com will be the base URL or domain, <i>/web</i> will be the first URL segment, <i>/shopping</i> will be the second URL segment, and <i>/login.jsp</i> will be the third URL segment.

Parameter	Description
	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 <i>/web/shopping</i>. Here, <i>/web</i> corresponds to the specification 1 (URL segment 1) and <i>/shopping</i> 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.eazycart.com/web/shopping/products.jsp, http://www.eazycart.com/web/shopping/products/travel/bags.jsp, http://www.eazycart.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 <i>/web/shopping</i>. This way, the test will create different page groups based on each of the second-level URL segments in the managed web site – eg., <i>/web/movies</i>, <i>/web/travel</i>, <i>/web/reservations</i>, <i>/partner/contacts</i> 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.eazycart.com/web/shopping/login.jsp, the test will report metrics for the descriptor <i>/shopping</i>. You can even set this parameter to 1,3. For a web site that contains URLs such as http://www.eazycart.com/web/shopping/products.jsp, http://www.eazycart.com/web/shopping/products/travel/bags.jsp, and http://www.eazycart.com/web/shopping/payment.jsp, this specification will result in the following descriptors: <i>/web/products</i>, <i>/web/products.jsp</i>, and <i>/web/payment.jsp</i>.
URL Patterns to be Ignored from Monitoring	By default, this test does not track requests to the following URL patterns: <i>*.js, *.css, *.jpeg, *.jpg, *.png</i>. 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 <i>.gif</i> and <i>.bmp</i> when monitoring, you need to alter the default specification as follows: <i>*.js, *.css, *.jpeg, *.jpg, *.png, *.gif, *.bmp</i> 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

Parameter	Description
	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 <i>none</i> 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: <i><Javascript error message>:<URL of the page/file where the message originated></i>. For instance, say that the <i>login.html</i> page in your web site runs a few Java scripts that throw <i>Object expected errors</i>, which you want the eG agent to ignore. In this case, your error specification can be as follows: <i>Object expected:http://www.eazykart.com/web/login.html</i>. Alternatively, you can provide only that text string with which the error message begins in your specification – eg., <i>Object:http://www.eazykart.com/web/login.html</i>. 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: <i>Object:login.html</i>. 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 <i>login.html</i> page points to a <i>validate.js</i> file, and you want to ignore the Object expected errors that this JS file throws, your error pattern specification will either be, <i>Object expected:validate.js</i>, or <i>Object:validate.js</i>. Multiple error message-URL combinations can also be provided as a comma-separated list. The format of your specification will be:

Parameter	Description
	<code><Javascript error message 1>:<Originating URL 1>,<Javascript error message 2>:<Originating URL 2>,...</code> For example, to ignore the <i>Object expected</i> and <i>Uncaught TypeError</i> errors in the <i>login.html</i> page, use the following specification: <i>Object:login.html,Uncaught:login.html</i> Likewise, to ignore the <i>Object expected</i> error in the <i>login.html</i> page and the <i>Uncaught TypeError</i> in the <i>validate.js</i> file, your specification will be: <i>Object:login.html,Uncaught:validate.js</i> If you want to ignore the <i>Uncaught TypeError</i> across all the pages of the web site, your specification will be as follows: <i>Uncaught TypeError:All</i> Note: When specifying the <code><Javascript error message></code> 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 <i>case-sensitive</i> pattern matching.
Include Page Type Categorization in Top-N DD	By default, the detailed diagnosis of this test reports the top-n transactions (in terms of <i>Avg page load time</i>) across page types. Accordingly, the Include Page Type Categorization in Top-N DD flag is set to No by default. If you want detailed diagnosis to report the top-n transactions per page type, then set this flag to Yes. In this case, if say the Maximum Healthy Transactions, Maximum Slow Transactions, and Maximum Error Transactions parameters are each set to 5, then the detailed diagnosis of this test will report the top-5 healthy, top-5 slow, and top-5 error transactions to Ajax pages, iFrames, Virtual pages, and base pages. As this may typically generate a large volume of data and increase the load on the eG database, you may want to set the 'Include Page Type Categorization in Top-N DD' to 'Yes', only if your eG database is sized sufficiently.
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, for each measurement period in the timeline chosen for detailed diagnosis. To identify the top 5, the eG agent sorts all healthy transactions in a

Parameter	Description
	measurement period 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 <i>0</i>. To view all healthy transactions, set the value of this parameter to <i>all</i>. Before setting the value of this parameter to <i>all</i>, make sure that you have a well-sized eG database in place, as this setting will store details of every transaction that registers an <i>Avg page load time</i> 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 <i>5</i> indicating that the detailed diagnosis of this test will display metrics related to the top-5 slow transactions, in terms of the <i>Average page load time</i>, for each measurement period in the timeline chosen for detailed diagnosis. To identify the top 5, the eG agent sorts all slow transactions in a measurement period in the descending order of their <i>Average page load time</i>, 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 <i>0</i>. To view all slow transactions, set the value of this parameter to <i>all</i>. Before setting the value of this parameter to <i>all</i>, make sure that you have a well-sized eG database in place, as this setting will store details of every transaction that registers an <i>Avg page load time</i> 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 <i>5</i>, indicating that the detailed diagnosis of this test will display metrics related to the top-5 transactions (per measurement period) that encountered JavaScript errors, based on when those errors occurred. To identify the top 5, the eG agent sorts all error transactions in a measurement period 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.

Parameter	Description
	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 <i>0</i>. To view all error transactions, set the value of this parameter to <i>all</i>. Before setting the value of this parameter to <i>all</i>, 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 <i>4000 millisecs</i> (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 <i>Maximum threshold</i> setting of the <i>Avg page load time</i> measure – i.e., both are set to <i>4000 millisecs</i> 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 <i>Avg page load time</i>. 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 Transaction Cutoff is changed to <i>6000 millisecs</i>, change the <i>Maximum Threshold</i> of the <i>Avg page load time</i> measure to <i>6000 millisecs</i> as well.
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, Virtual Page, and/or iFrame views as well in the dashboard, by selecting the relevant options from this list box.
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 <i>0</i> - 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

Parameter	Description
	manager, despite the fact that their value is 0. These zero values will clearly indicate to users that there is no traffic to the monitored web site/web application.
DD Frequency	Refers to the frequency with which detailed diagnosis measures are to be generated for this test. The default is 3:1. This indicates that, by default, detailed measures will be generated every third 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 <i>none</i> 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.

Measurement	Description	Measurement Unit	Interpretation
			If the eG agent monitoring the <i>Real User Monitor</i> 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 <i>Page views</i> 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.

Measurement	Description	Measurement Unit	Interpretation
			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 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 could be the country that is causing the slowness. You may want to compare the values of the of the <i>Average browser initial request time</i>, <i>Average network time</i>, <i>Average server time</i>, <i>Average content download time</i>, and <i>Average browser</i>

Measurement	Description	Measurement Unit	Interpretation
			<i>render time</i> measures for that country to know what is exactly causing users from that country to experience slowness. If the <i>Average browser initial request time</i> is the highest, it can be attributed to a slowdown in redirection, AppCaching, or because the request spent too much time waiting on the browser. If the <i>Average network time</i> is the highest, it denotes that the network is the problem source. This in turn can be caused by TCP connection latencies, delays in domain look up, and slowdown in SSL handshake. If the <i>Average content download time</i> measure reports an abnormally high value, then it could be owing to slowness in downloading the HTML document or in building the DOM. On the other hand, if the <i>Average server time</i> 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. To know which pages accessed from this country are slow, use the detailed diagnosis of this measure.
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	Indicates the percentage	Percent	The value of this measure indicates the

Measurement	Description	Measurement Unit	Interpretation
percentage	of page views that delivered a satisfactory experience to users from this country.		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 experience of users from this country has not been up to the mark. A value less than 50% is indicative of a serious problem, where most of the page view requests from this country are either slow or have encountered Javascript errors. Under such circumstances, to know what exactly is affecting the experience of users from this country, compare the value of the <i>Slow page view percentage</i> with that of the <i>Javascript error page view percentage</i> for that country. 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 <i>Slow page views</i> measure to know which pages accessed by users from this country are slow and where these pages are losing time. If JavaScript errors are the problem, use the detailed diagnosis of the <i>JavaScript error view percentage</i> measure to know what errors occurred in which pages accessed from this country.
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 <i>Slow page views</i> measure to identify the slow pages and isolate the root-cause of the slowness – is

Measurement	Description	Measurement Unit	Interpretation
			it the front end? the network? or the backend?
JavaScript error view percentage	Indicates the percentage of page views from this country 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 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 frequently experienced slowness when accessing pages. To know which pages were slow when accessed from this country 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 from this country. 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.

Measurement	Description	Measurement Unit	Interpretation
			Ideally, the value of this measure should be the same as that of the <i>Page views</i> 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 <i>Tolerating page views</i> and the <i>Frustrated page views</i>, it is a clear indicator that the experience of users from this country has been below-par. In such a case, use the detailed diagnosis of the <i>Tolerating page views</i> and <i>Frustrated page views</i> 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 <i>Average page load time</i> of a page exceeds the Slow Transaction Cutoff configuration of this test, but is less than 4 times the Slow Transaction Cutoff (i.e., $< 4 * \text{Slow Transaction Cutoff}$), then such a page view is considered to be a <i>Tolerating page view</i>. Ideally, the value of this measure should be 0. A value higher than that of the <i>Satisfied page views</i> measure is a cause for concern, as it implies that the overall experience of users from 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 <i>Average page load time</i> of a page is over 4 times the Slow Transaction Cutoff configuration of this test (i.e., $> 4 * \text{Slow Transaction Cutoff}$), then such a page view is considered to be a <i>Frustrated page view</i>.

Measurement	Description	Measurement Unit	Interpretation
			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 <i>Satisfied page views</i> measure is a cause for concern, as it implies that the experience of users from this country 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 browser initial request time	Indicates the interval between when a request was received from users in this country, when it followed a redirect, to when it was processed in the AppCache.	ms	In Figure 3.205, the <i>Average browser initial request time</i> is the time spent between the navigationStart event to the domainLookupStart event. This also includes the time spent by the browser waiting for one event to end and the next to begin. In short, this measure is the sum of <i>Average redirect time</i>, <i>Average AppCache time</i>, and <i>Average browser wait time</i> measures. This means that if a request takes too long to follow a redirection, or if the AppCache takes too long to process the request, or if the request spends too much time on the browser for the previous request to complete, then the value of this measure will significantly increase. This in turn will impact user experience, and consequently, the <i>Apdex score</i>. This is why, if this measure reports an abnormal value for a country, it is important that you compare the value of

Measurement	Description	Measurement Unit	Interpretation
			the <i>Average redirect time</i> , <i>Average AppCache time</i> , and <i>Average browser wait time</i> measures for the country, to figure out where the request spent maximum time - in redirection? in the AppCache? or when waiting on the browser?
Average server time	Indicates the interval between the start of processing of a request from this country on the browser to when response is received.	ms	In Figure 3.205, the <i>Average server time</i> is the time spent between the <i>requestStart</i> event and <i>responseStart</i> event. Ideally, a low value is desired for this measure, as high values will certainly hurt the <i>Apdex score</i> 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 <i>Average server time</i> will increase, leaving users with an unsatisfactory experience with the web site/web application. Note: This test uses the Navigation Timing API to measure web site performance. The Navigation Timing API typically exposes several properties that offer information about the time at which different page load events happen (see Figure 3.205) - eg., the

Measurement	Description	Measurement Unit	Interpretation
			requestStart event, the responseStart event, etc. This test uses the time stamps provided by the Navigation Timing API to compute and report the duration of page load events, so you can accurately identify where page loading is bottlenecked. The Navigation Timing API on Internet Explorer (IE) v11 reports an incorrect time stamp for the <i>requestStart</i> event of the page loading process (see Figure 3.205). As a result, for page view requests initiated from IE 11 browsers alone, eG RUM will report incorrect values for this measure. This issue was noticed in IE 11 in April 2019. It is recommended that you track hot fixes/patches released by Microsoft post April 2019, study the release notes of such fixes/patches, and determine if this bug has been fixed in any. If so, then you are advised to apply that fix/patch on IE 11 to resolve the issue. Until then, we recommend that you use the following workaround to accurately measure the <i>Average server time</i> of a page view request. - Deploy eG BTM (Java/.NET, as the case may be) on the backend server hosting the target web site/web application. - Use eG BTM to trace the path of transactions to the target web site/web application and enable the (Java or .NET) Business Transactions test to capture metrics on transaction performance.

Measurement	Description	Measurement Unit	Interpretation
			- Next, use the detailed diagnostics reported by eG RUM to identify the page view requests coming from the IE 11 browser. Make a note of the values in the Request time, URL and Query Params columns of detailed diagnosis. - Then, search the detailed diagnostics of the (Java or .NET) Business Transactions test for transactions with the same URL, Request time, and Query Params as reported by eG RUM. - The response time that eG BTM reports for each of these transactions is the <i>Average server time</i> of those transactions. Note that this workaround applies only for those transaction URLs that are captured and reported as part of detailed diagnostics.
Average network time	Indicates the elapsed time since a user from this country initiates a request to the web site/application and the start of fetching the response document from it.	ms	In Figure 3.205, the time spent between <i>navigationStart</i> and <i>requestStart</i> makes up the <i>Average network time</i>. This includes the time to perform DNS lookups, the time to establish a TCP connection with the server, and the time to perform an SSL handshake. In other words, the value of this measure is nothing but the sum of the <i>Average DNS lookup time</i>, the <i>Average TCP connection time</i>, and the <i>Average SSL handshake time</i> measures. Ideally, the value of this measure should be low. A very high value will often end up

Measurement	Description	Measurement Unit	Interpretation
			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 <i>Average DNS lookup time</i> , <i>Average TCP connection time</i>, and <i>Average SSL handshake time</i> measures to know to what this delay can be attributed – a delay in domain name resolution? Or a poor network connection to the server? or slowness in SSL negotiations between the browser and the server? Note: This test uses the Navigation Timing API to measure web site performance. The Navigation Timing API typically exposes several properties that offer information about the time at which different page load events happen (see Figure 3.205) - eg., the <i>requestStart</i> event, the <i>responseStart</i> event, etc. This test uses the time stamps provided by the Navigation Timing API to compute and report the duration of page load events, so you can accurately identify where page loading is bottlenecked. The Navigation Timing API on Internet Explorer (IE) v11 reports an incorrect time stamp for the <i>requestStart</i> event of the page loading process (see 3.6.6). As a result, for page view requests initiated from IE 11 browsers alone, eG RUM will report incorrect values for this measure. This issue was noticed in IE 11 in April 2019. It is recommended that you track hot fixes/patches released by Microsoft post April 2019, study the release notes of such fixes/patches, and determine if this bug has been fixed in any. If so, then you are

Measurement	Description	Measurement Unit	Interpretation
			advised to apply that fix/patch on IE 11 to resolve the issue.
Average content download time	Indicates the time to make the complete HTML document (DOM) available for JavaScript to apply rendering logic on the pages accessed from this country.	ms	This is the time spent between the responseStart event (shown in 3.6.6) and the documentContentLoadedEventStart event (not shown in 3.6.6). The documentContentLoadedEventStart is typically fired just before the domContentLoaded event, which is just after browser has finished downloading and parsing all the scripts that had defer set and no async attribute. In summary, the <i>Average content download time</i> measure is the sum of the values of the <i>Average DOM download time</i> and <i>Average DOM processing time</i> measures. This means that an abnormal increase in any of the above-mentioned time values will increase the value of this measure. If content downloading takes unusually long for a request, then you must compare the values of the <i>Average DOM download time</i> and <i>Average DOM processing time</i> measures to figure out what is causing the delay - is it because of the poor responsiveness of server, cache, or local resource? or is it because DOM processing took too long?
Average browser render time	Indicates the time taken to complete the download of remaining resources, including images, and to finish rendering the pages accessed from this country.	ms	A high value of this measure indicates that page rendering is taking too long. This can be attributed to an in-optimal HTML document architecture, complex CSS' (eg., deeply nested rules, slow selectors, complicated effects such as round borders, gradients etc.), and large sized images.

Measurement	Description	Measurement Unit	Interpretation
			If the <i>Average page load time</i> measure reports an abnormally high value, then you may want to compare the value of this measure with that of the <i>Average browser initial request time</i> , <i>Average network time</i> , <i>Average server time</i> , and <i>Average content download time</i> measures to nail the exact source of the bottleneck.
Average DOM download time	Indicates the time taken to download the complete HTML document for page view requests from this country, on the browser.	ms	The value of this measure is the time that elapsed between the responseStart and responseEnd events in 3.6.6. 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 accessed from 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. Consequently, content download will be delayed, thus adversely impacting user experience with the web site/web application. Ideally therefore, the value of this measure should be low.
Average browser wait time	Indicates the time that requests from this country spent on the browser, waiting for another request to complete.	ms	This is the sum of the time between every two consecutive events, starting with the navigationStart event till the requestStart event in 3.6.6. Typically, web browsers limit the number of active connections for each domain. Most modern browsers (eg., Chrome) support only six simultaneous requests/connections. In this case therefore, when the seventh request

Measurement	Description	Measurement Unit	Interpretation
			comes in, that request waits on the browser until the six requests sent previously are processed. The waiting time of the seventh request is the browser wait time. High browser wait time can prolong the browser's initial request time, thus adversely impacting the overall responsiveness of the web site/application. This is why, if the <i>Average browser initial request time</i> measure reports an abnormally high value, you will have to compare the values of the <i>Average browser wait time</i>, <i>Average redirection time</i>, and <i>Average AppCache time</i> measure to determine whether/not the initial request delay observed on the browser is because requests have been waiting on the browser for too long a time. Some of the means by which you can reduce browser waits are briefly discussed below: • Browsers such as Mozilla Firefox support upto 10 parallel requests. You may want to recommend such browsers for your web site/web application users, so that more requests are processed and fewer requests are queued on the browser, thus reducing browser wait time. • Web site/application developers can try domain sharding - i.e., split content across multiple domains. Typically,

Measurement	Description	Measurement Unit	Interpretation
			when a user connects to a web page, his or her browser scans the resulting HTML for resources to download. Normally these resources are supplied by a single domain - the domain providing the web page or a domain created specifically for resources. With domain sharding, the user's browser connects to two or more different domains to simultaneously download the resources needed to render the web page. This allows website/application to be delivered faster to users as they do not have to wait for the previous set of requests to end before beginning the next set.
Average redirection time	Indicates the time that requests from this country spent in redirection before fetching the pages.	ms	In 3.6.6, this is the elapsed time between the redirectStart and redirectEnd events. URL redirection, also known as URL forwarding, is a technique to give a page, a form, or a whole web site/application, more than one URL address. Usually, web site administrators use URL redirection to: • Redirect users to the mobile version of the site • Redirect users to secured pages • Redirect users to the latest version of the resource/content • Redirect users to pages specific to their geo location

Measurement	Description	Measurement Unit	Interpretation
			- Redirecting canonical URLs Though redirects are useful, they have to be kept at a minimum, as each redirect on a page adds latency to the overall page load time. This is because, when a user enters a domain into the browser and hits enter, the DNS resolution process is triggered and the domain is resolved to its corresponding IP address in a few milliseconds. If the landing page has another redirect, then the browser repeats the entire DNS resolution process once again to guide the user to the correct web page. The multiple redirect requests are taxing on the browser resources and slow down the page load. Web page load time is also affected by internal redirects; for example, if the page tries to load content from a URL that has been redirected to newer or updated content, then the browser must create additional requests to fetch the valid content. These redirects result in additional round trips between the browser and the web server which pushes the load time higher; the perceived performance is degraded every time the browser encounters a redundant redirect. Web site/application performance is also impacted if redirects are not implemented correctly. Some of the common redirect errors are: - Multiple redirects: The higher the number of redirects on a page, the higher is its page load time.

Measurement	Description	Measurement Unit	Interpretation
			- Invalid redirects: There are often instances where the web site administrator sets up bulk redirects without verifying the validity of the redirects. The site may also have old redirects that were never cleaned up. This can cause several issues on the site like broken links and 404s. - Redirect loop: When there are several redirects on the page that are linked to each other, it creates a chain of redirects which may loop back to the same URL that initiated the redirect. This results in a redirect loop error and the user will not be able to access the site. Therefore, if you find that the value of the <i>Average page load time</i> measure is abnormally high owing to an unusually high value for the <i>Average redirection time</i> measure, then make sure you follow the best practices outlined below while implementing redirects, so you can significantly reduce page load time and improve user experience: - Avoid redundant redirects: It's recommended to avoid redirects where possible and to use this method only when absolutely needed. This will cut down unnecessary overhead and improve the perceived performance of the page.

Measurement	Description	Measurement Unit	Interpretation
			- Chain redirects: When a URL is linked to another URL, this creates a chained redirect. Each URL added to the chain adds latency to the page. Chained redirects have a negative impact not only on page speed, but also SEO. - Clean-up redirects: You may have hundreds of redirects on your website and it could be one of the main factors affecting page speed. Old redirects may conflict with new URLs, backlinks can cause odd errors on the page. It is recommended to verify all the redirects you have set up and to remove the ones that are no longer needed. Retain the old links that have major referral traffic, while those that are rarely accessed can be removed. This exercise will help improve page speed significantly.
Average AppCache time	Indicates the time taken to check whether/not the requests from this country can be serviced by the AppCache.	ms	HTML5 provides an application caching mechanism that lets web-based applications run offline. Developers can use the Application Cache (AppCache) interface to specify resources that the browser should cache and make available to offline users. Applications that are cached load and work correctly even if users click the refresh button when they are offline. Using an application cache gives an application the following benefits: - Offline browsing: users can navigate a site even when they are offline. - Speed: cached resources are local, and therefore load faster.

Measurement	Description	Measurement Unit	Interpretation
			Reduced server load: the browser only downloads resources that have changed from the server. To enable the application cache for an application, you must include the manifest attribute in the <html> element in your application's pages. The manifest attribute references a cache manifest file, which is a text file that lists resources (files) that the browser should cache for your application. The browser does not cache pages that do not contain the manifest attribute, unless such pages are explicitly listed in the manifest file itself. You do not need to list all the pages you want cached in the manifest file, the browser implicitly adds every page that the user visits and that has the manifest attribute set to the application cache. When the browser visits a document that includes the manifest attribute, if no application cache exists, the browser loads the document and then fetches all the entries listed in the manifest file, creating the first version of the application cache. Subsequent visits to that document cause the browser to load the document and other assets specified in the manifest file from the application cache. If the manifest file has changed, all the files listed in the manifest—as well as those already added to the cache—are fetched into a temporary cache. Once all the files have been successfully retrieved, they are moved into the real

Measurement	Description	Measurement Unit	Interpretation
			offline cache automatically. Since the document has already been loaded into the browser from the cache, the updated document will not be rendered until the document is reloaded (either manually or programmatically). A high value of this measure signifies that requests are spending too much time in the AppCache. This also introduces page loading latencies, which have an adverse effect on user-perceived performance of a web site/application. Common reasons for AppCaching issues and their practical solutions are detailed below: • If the media type is not set, then AppCache will not work. To avoid this, make sure that the manifest file is always served under the correct media type of text/cache-manifest. • If the manifest file is not served to the web browser from the same origin as the host page, the manifest file will fail to load. To avoid this, make sure that the manifest file is always served from the same origin as the host page. However, note that the manifest file can hold reference to resources to be cached from other domains. • The relative URLs that you mention in the manifest are relative to the manifest file and not to the document where you reference the manifest file. If you make this error when the manifest and the

Measurement	Description	Measurement Unit	Interpretation
			reference are not in the same path, the resources will fail to load and in turn the manifest file will not be loaded. This will stall appcaching. Any change made to the manifest file will cause the entire set of files to be downloaded again. Moreover, if a manifest file is added to a HTML file, it forces all resources to be downloaded synchronously as soon as the manifest file is downloaded. As a result, resources that may not yet be required, such as JavaScript or an image below the fold, will be downloaded at the start of the page. This can increase page load time significantly. The solution to this is to load the Application Cache from a simple HTML file loaded in an iframe. This not only avoids caching dynamic HTML but also avoids the Application Cache being downloaded asynchronously after the page load has completed.
Average DNS lookup time	Indicates the time taken by the browser to perform the domain lookup for connecting to the pages requested by this country.	ms	A high value for this measure will not only affect DNS lookup, but will also impact the <i>Average network time</i> and <i>Average page load time</i> 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 for accessing pages from this country.	ms	A bad network connection between the browser client and the server can delay TCP connections to the server. As a result, the <i>Average network time</i> too will increase, thus impacting page load time and overall user experience with the web site/web application.

Measurement	Description	Measurement Unit	Interpretation
Average SSL handshake time	Indicates the time taken by requests from this country to complete the SSL handshake.	ms	An SSL handshake happens when a browser makes a secure request for content, also known as an encrypted HTTPS connection. The user's browser and server negotiate encrypted keys and certificates to establish a secure connection between each other. Because this SSL negotiation requires exchanges between the browser and your server, it increases the time spent by the request on the network (i.e., it adds to the value of the Average network time measure). This in turn increases page load time. In fact, an SSL handshake, along with DNS lookup and TCP handshake add three round trips to the page load time. A quick fix to this is to use HTTP/2. HTTP/2 can use caching to reduce SSL setup to only one round trip.

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

Configurable parameters for the test

Parameter	Description
Test Period	How often should the test be executed.
Proxy Host	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 <i>none</i> , indicating that the eG agent does not communicate with the collector via a proxy.
Proxy Port	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 <i>none</i> , indicating that the eG agent does not communicate with the collector via a proxy.
Proxy Username and Proxy Password	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 Proxy Username and Proxy Password . If no proxy server is used or if the proxy server used does not require authentication, then set the ProxyUsername and ProxyPassword to <i>none</i> .
Confirm Password	Confirm the Proxy Password by retyping it here.
	Note: If you Reconfigure the test later to change the values of the Proxy Username , Proxy Password , 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.

MAXIMUM ALLOWED PAGE VISITS PER DAY

100000

Parameter	Description
	Configuring the number of allowed page visits By default, the Maximum allowed page visits per day is set to <i>100000</i>. This implies that the test will consider only the first <i>100000</i> requests in a day for monitoring. All page visits beyond <i>100000</i> will by default be excluded from the test's monitoring purview. You can increase or decrease this limit, if you so need.
Consider JavaScript errors in Apex calculations	The formula for computing the Apex score is as follows: $[Satisfied\ requests + (Tolerating\ requests/2)]/Total\ samples$ By default, the count of <i>Satisfied</i> and <i>Tolerating requests</i> used for Apex score computation includes the count of requests that experienced JavaScript errors. This is why, the Consider JavaScript errors in Apex calculations flag is set to Yes by default. Since JavaScript errors also impact user experience, this default setting will result in an Apex score that presents a near-real picture of the performance of a web site/web application. If you do not want the Apex score to be tainted by error requests, then set this flag to <i>No</i>. In this case, requests with JavaScript errors will not be considered for Apex calculations, even if such requests have a response time that is well-within (satisfied) or in violation (tolerating) of the Slow Transaction Cutoff.
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.eazycart.com/web/shopping/login.jsp, then http://www.eazycart.com will be the base URL or domain, <i>/web</i> will be the first URL segment, <i>/shopping</i> will be the second URL segment, and <i>/login.jsp</i> 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 <i>/web/shopping</i>. Here, <i>/web</i> corresponds to the specification 1 (URL segment 1) and <i>/shopping</i> 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.eazycart.com/web/shopping/products.jsp, http://www.eazycart.com/web/shopping/products/travel/bags.jsp, http://www.eazycart.com/web/shopping/payment.jsp, etc., then the eG agent will

Parameter	Description
	track the requests to and responses from all these web pages, aggregate the results, and present the aggregated metrics for the descriptor <i>/web/shopping</i>. This way, the test will create different page groups based on each of the second-level URL segments in the managed web site – eg., <i>/web/movies</i>, <i>/web/travel</i>, <i>/web/reservations</i>, <i>/partner/contacts</i> 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.eazycart.com/web/shopping/login.jsp, the test will report metrics for the descriptor <i>/shopping</i>. You can even set this parameter to 1,3. For a web site that contains URLs such as http://www.eazycart.com/web/shopping/products.jsp, http://www.eazycart.com/web/shopping/products/travel/bags.jsp, and http://www.eazycart.com/web/shopping/payment.jsp, this specification will result in the following descriptors: <i>/web/products</i>, <i>/web/products.jsp</i>, and <i>/web/payment.jsp</i>.
URL Patterns to be Ignored from Monitoring	By default, this test does not track requests to the following URL patterns: <i>*.js, *.css, *.jpeg, *.jpg, *.png</i>. 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 <i>.gif</i> and <i>.bmp</i> when monitoring, you need to alter the default specification as follows: <i>*.js, *.css, *.jpeg, *.jpg, *.png, *.gif, *.bmp</i> 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.

Parameter	Description
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 <i>none</i> 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: <i><Javascript error message>:<URL of the page/file where the message originated></i>. For instance, say that the <i>login.html</i> page in your web site runs a few Java scripts that throw <i>Object expected errors</i>, which you want the eG agent to ignore. In this case, your error specification can be as follows: <i>Object expected:http://www.eazycart.com/web/login.html</i>. Alternatively, you can provide only that text string with which the error message begins in your specification – eg., <i>Object:http://www.eazycart.com/web/login.html</i>. 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: <i>Object:login.html</i>. 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 <i>login.html</i> page points to a <i>validate.js</i> file, and you want to ignore the <i>Object expected errors</i> that this JS file throws, your error pattern specification will either be, <i>Object expected:validate.js</i>, or <i>Object:validate.js</i>. Multiple error message-URL combinations can also be provided as a comma-separated list. The format of your specification will be: <i><Javascript error message 1>:<Originating URL 1>,<Javascript error message 2>:<Originating URL 2>,. . .</i> For example, to ignore the <i>Object expected</i> and <i>Uncaught TypeError</i> errors in the <i>login.html</i> page, use the following specification: <i>Object:login.html,Uncaught:login.html</i> Likewise, to ignore the <i>Object expected</i> error in the <i>login.html</i> page and the <i>Uncaught TypeError</i> in the <i>validate.js</i> file, your specification will be: <i>Object:login.html,Uncaught:validate.js</i> If you want to ignore the <i>Uncaught TypeError</i> across all the pages of the web site,

Parameter	Description
	your specification will be as follows: <i>Uncaught TypeError:All</i> Note: When specifying the <i><Javascript error message></i> 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 <i>case-sensitive</i> pattern matching.
Include Page Type Categorization in Top-N DD	By default, the detailed diagnosis of this test reports the top-n transactions (in terms of <i>Avg page load time</i>) across page types. Accordingly, the Include Page Type Categorization in Top-N DD flag is set to No by default. If you want detailed diagnosis to report the top-n transactions per page type, then set this flag to Yes. In this case, if say the Maximum Healthy Transactions, Maximum Slow Transactions, and Maximum Error Transactions parameters are each set to 5, then the detailed diagnosis of this test will report the top-5 healthy, top-5 slow, and top-5 error transactions to Ajax pages, iFrames, Virtual pages, and base pages. As this may typically generate a large volume of data and increase the load on the eG database, you may want to set the 'Include Page Type Categorization in Top-N DD' to 'Yes', only if your eG database is sized sufficiently.
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, for each measurement period in the timeline chosen for detailed diagnosis. To identify the top 5, the eG agent sorts all healthy transactions in a measurement period 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 <i>all</i>. Before setting the value of this parameter to <i>all</i>, make sure that you have a well-sized eG database in place, as this setting will store details of every

Parameter	Description
	transaction that registers an <i>Avg page load time</i> 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 the top-5 slow transactions, in terms of the <i>Average page load time</i>, for each measurement period in the timeline chosen for detailed diagnosis. To identify the top 5, the eG agent sorts all slow transactions in a measurement period in the descending order of their <i>Average page load time</i>, 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 <i>all</i>. Before setting the value of this parameter to <i>all</i>, make sure that you have a well-sized eG database in place, as this setting will store details of every transaction that registers an <i>Avg page load time</i> 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 (per measurement period) that encountered JavaScript errors, based on when those errors occurred. To identify the top 5, the eG agent sorts all error transactions in a measurement period 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 <i>all</i>. Before setting the value of this parameter to <i>all</i>, 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

Parameter	Description
Cutoff (Ms)	slow. To determine whether/not a page is slow, this test uses the Slow Transaction Cutoff parameter. By default, this parameter is set to <i>4000 millisecs</i> (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 <i>Maximum threshold</i> setting of the <i>Avg page load time</i> measure – i.e., both are set to <i>4000 millisecs</i> 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 <i>Avg page load time</i>. 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 Transaction Cutoff is changed to <i>6000 millisecs</i>, change the <i>Maximum Threshold</i> of the <i>Avg page load time</i> measure to <i>6000 millisecs</i> as well.
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, Virtual Page, and/or iFrame views as well in the dashboard, by selecting the relevant options from this list box.
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 <i>0</i> - 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 <i>0</i>. These zero values will clearly indicate to users that there is no traffic to the monitored web site/web application.
DD Frequency	Refers to the frequency with which detailed diagnosis measures are to be generated for this test. The default is <i>3:1</i> . This indicates that, by default, detailed measures will be generated every third 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 <i>none</i> 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

Parameter	Description
	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 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 <i>Real User Monitor</i> 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

Measurement	Description	Measurement Unit	Interpretation
			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 <i>Page views</i> 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 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.

Measurement	Description	Measurement Unit	Interpretation
			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 time	Indicates the average time that pages accessed by users from this city 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 cities, you will be able to tell if the page load time is significantly higher for any one city – this could be the city that is causing the slowness. You may want to compare the values of the of the <i>Average browser initial request time</i>, <i>Average network time</i>, <i>Average server time</i>, <i>Average content download time</i>, and <i>Average browser render time</i> measures for that city to know what is exactly causing users from that city to experience slowness. If the <i>Average browser initial request time</i> is the highest, it can be attributed to a slowdown in redirection, AppCaching, or because the request spent too much time waiting on the browser. If the <i>Average network time</i> is the highest, it denotes

Measurement	Description	Measurement Unit	Interpretation
			that the network is the problem source. This in turn can be caused by TCP connection latencies, delays in domain look up, and slowdown in SSL handshake. If the <i>Average content download time</i> measure reports an abnormally high value, then it could be owing to slowness in downloading the HTML document or in building the DOM. On the other hand, if the <i>Average server time</i> 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. To know which pages accessed from this city are slow, use the detailed diagnosis of this measure.
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 experience of users from this city has not been up to the mark. A value less than 50% is indicative

Measurement	Description	Measurement Unit	Interpretation
			of a serious problem, where most of the page view requests from this city are either slow or have encountered Javascript errors. Under such circumstances, to know what exactly is affecting the experience of users from this city, compare the value of the <i>Slow page view percentage</i> with that of the <i>Javascript error page view percentage</i> for that city. 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 <i>Slow page views</i> measure to know which pages accessed by users from this city are slow and where these pages are losing time. If JavaScript errors are the problem, use the detailed diagnosis of the <i>JavaScript error view percentage</i> measure to know what errors occurred in which pages accessed from this city.
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 <i>Slow page views</i> 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 from this city 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	Indicates the number of	Number	A page view is considered to be slow when

Measurement	Description	Measurement Unit	Interpretation
(Tolerating & Frustrated)	slow page views from this city.		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. To know which pages were slow when accessed from this city 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 from this city. 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 <i>Page views</i> 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 <i>Tolerating page views</i> and the <i>Frustrated page views</i>, it

Measurement	Description	Measurement Unit	Interpretation
			is a clear indicator that the experience of users from this city has been below-par. In such a case, use the detailed diagnosis of the <i>Tolerating page views</i> and <i>Frustrated page views</i> 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 <i>Average page load time</i> of a page exceeds the Slow Transaction Cutoff configuration of this test, but is less than 4 times the Slow Transaction Cutoff (i.e., $< 4 * \text{Slow Transaction Cutoff}$), then such a page view is considered to be a <i>Tolerating page view</i>. Ideally, the value of this measure should be 0. A value higher than that of the <i>Satisfied page views</i> measure is a cause for concern, as it implies that the overall experience of users from 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 <i>Average page load time</i> of a page is over 4 times the Slow Transaction Cutoff configuration of this test (i.e., $> 4 * \text{Slow Transaction Cutoff}$), 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 <i>Satisfied page views</i> measure is a cause for concern, as it implies that the experience of users from this city has been less than satisfactory. To know which

Measurement	Description	Measurement Unit	Interpretation
			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 browser initial request time	Indicates the interval between when a request was received from users in this city, when it followed a redirect, to when it was processed in the AppCache.	ms	In Figure 3.205, the <i>Average browser initial request time</i> is the time spent between the <i>navigationStart</i> event to the <i>domainLookupStart</i> event. This also includes the time spent by the browser waiting for one event to end and the next to begin. In short, this measure is the sum of <i>Average redirect time</i>, <i>Average AppCache time</i>, and <i>Average browser wait time</i> measures. This means that if a request takes too long to follow a redirection, or if the AppCache takes too long to process the request, or if the request spends too much time on the browser for the previous request to complete, then the value of this measure will significantly increase. This in turn will impact user experience, and consequently, the <i>Apdex score</i>. This is why, if this measure reports an abnormal value for a city, it is important that you compare the value of the <i>Average redirect time</i>, <i>Average AppCache time</i>, and <i>Average browser wait time</i> measures for the city, to figure out where the request spent maximum time - in redirection? in the AppCache? or when waiting on the browser?
Average server time	Indicates the interval between the start of processing of a request	ms	In Figure 3.205, the <i>Average server time</i> is the time spent between the <i>requestStart</i> event and <i>responseStart</i>

Measurement	Description	Measurement Unit	Interpretation
	from this city on the browser to when response is received.		event. Ideally, a low value is desired for this measure, as high values will certainly hurt the <i>Apdex score</i> 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 <i>Average server time</i> will increase, leaving users with an unsatisfactory experience with the web site/web application. Note: This test uses the Navigation Timing API to measure web site performance. The Navigation Timing API typically exposes several properties that offer information about the time at which different page load events happen (see Figure 3.205) - eg., the <i>requestStart</i> event, the <i>responseStart</i> event, etc. This test uses the time stamps provided by the Navigation Timing API to compute and report the duration of page load events, so you can accurately identify where page loading is bottlenecked. The Navigation Timing API on Internet Explorer (IE) v11 reports an incorrect time stamp for the <i>requestStart</i> event of the page loading process (see Figure 3.205).

Measurement	Description	Measurement Unit	Interpretation
			As a result, for page view requests initiated from IE 11 browsers alone, eG RUM will report incorrect values for this measure. This issue was noticed in IE 11 in April 2019. It is recommended that you track hot fixes/patches released by Microsoft post April 2019, study the release notes of such fixes/patches, and determine if this bug has been fixed in any. If so, then you are advised to apply that fix/patch on IE 11 to resolve the issue. Until then, we recommend that you use the following workaround to accurately measure the <i>Average server time</i> of a page view request. - Deploy eG BTM (Java/.NET, as the case may be) on the backend server hosting the target web site/web application. - Use eG BTM to trace the path of transactions to the target web site/web application and enable the (Java or .NET) Business Transactions test to capture metrics on transaction performance. - Next, use the detailed diagnostics reported by eG RUM to identify the page view requests coming from the IE 11 browser. Make a note of the values in the Request time, URL and Query Params columns of detailed diagnosis. - Then, search the detailed diagnostics of the (Java or .NET)

Measurement	Description	Measurement Unit	Interpretation
			Business Transactions test for transactions with the same URL, Request time, and Query Params as reported by eG RUM. ◦ The response time that eG BTM reports for each of these transactions is the <i>Average server time</i> of those transactions. Note that this workaround applies only for those transaction URLs that are captured and reported as part of detailed diagnostics.
Average network time	Indicates the elapsed time since a user from this city initiates a request to the web site/application and the start of fetching the response document from it.	ms	In Figure 3.205, the time spent between <i>navigationStart</i> and <i>requestStart</i> makes up the <i>Average network time</i>. This includes the time to perform DNS lookups, the time to establish a TCP connection with the server, and the time to perform an SSL handshake. In other words, the value of this measure is nothing but the sum of the <i>Average DNS lookup time</i>, the <i>Average TCP connection time</i>, and the <i>Average SSL handshake time</i> 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 <i>Average DNS lookup time</i>, <i>Average TCP connection time</i>, and <i>Average SSL handshake time</i> measures to know to what this delay can be attributed – a delay in domain name resolution? Or a poor network connection to the server? or slowness in SSL negotiations between the

Measurement	Description	Measurement Unit	Interpretation
			browser and the server? Note: This test uses the Navigation Timing API to measure web site performance. The Navigation Timing API typically exposes several properties that offer information about the time at which different page load events happen (see Figure 3.205) - eg., the <code>requestStart</code> event, the <code>responseStart</code> event, etc. This test uses the time stamps provided by the Navigation Timing API to compute and report the duration of page load events, so you can accurately identify where page loading is bottlenecked. The Navigation Timing API on Internet Explorer (IE) v11 reports an incorrect time stamp for the <i>requestStart</i> event of the page loading process (see 3.6.6). As a result, for page view requests initiated from IE 11 browsers alone, eG RUM will report incorrect values for this measure. This issue was noticed in IE 11 in April 2019. It is recommended that you track hot fixes/patches released by Microsoft post April 2019, study the release notes of such fixes/patches, and determine if this bug has been fixed in any. If so, then you are advised to apply that fix/patch on IE 11 to resolve the issue.
Average content download time	Indicates the time to make the complete HTML document (DOM) available for JavaScript to apply rendering logic on the pages accessed from this city.	ms	This is the time spent between the <code>responseStart</code> event (shown in 3.6.6) and the <code>documentContentLoadedEventStart</code> event (not shown in 3.6.6). The <code>documentContentLoadedEventStart</code> is typically fired just before the <code>domContentLoaded</code> event, which is just after browser has finished downloading and parsing all the scripts that had defer

Measurement	Description	Measurement Unit	Interpretation
			set and no async attribute. In summary, the <i>Average content download time</i> measure is the sum of the values of the <i>Average DOM download time</i> and <i>Average DOM processing time</i> measures. This means that an abnormal increase in any of the above-mentioned time values will increase the value of this measure. If content downloading takes unusually long for a request, then you must compare the values of the <i>Average DOM download time</i> and <i>Average DOM processing time</i> measures to figure out what is causing the delay - is it because of the poor responsiveness of server, cache, or local resource? or is it because DOM processing took too long?
Average browser render time	Indicates the time taken to complete the download of remaining resources, including images, and to finish rendering the pages accessed from this city.	ms	A high value of this measure indicates that page rendering is taking too long. This can be attributed to an in-optimal HTML document architecture, complex CSS' (eg., deeply nested rules, slow selectors, complicated effects such as round borders, gradients etc.), and large sized images. If the <i>Average page load time</i> measure reports an abnormally high value, then you may want to compare the value of this measure with that of the <i>Average browser initial request time</i>, <i>Average network time</i>, <i>Average server time</i>, and <i>Average content download time</i> measures to nail the exact source of the bottleneck.
Average DOM download time	Indicates the time taken to download the	ms	The value of this measure is the time that elapsed between the responseStart and

Measurement	Description	Measurement Unit	Interpretation
	complete HTML document for page view requests from this city, on the browser.		responseEnd events in 3.6.6. 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 accessed from 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. Consequently, content download will be delayed, thus adversely impacting user experience with the web site/web application. Ideally therefore, the value of this measure should be low.
Average browser wait time	Indicates the time that requests from this city spent on the browser, waiting for another request to complete.	ms	This is the sum of the time between every two consecutive events, starting with the navigationStart event till the requestStart event in 3.6.6. Typically, web browsers limit the number of active connections for each domain. Most modern browsers (eg., Chrome) support only six simultaneous requests/connections. In this case therefore, when the seventh request comes in, that request waits on the browser until the six requests sent previously are processed. The waiting time of the seventh request is the browser wait time. High browser wait time can prolong the browser's initial request time, thus adversely impacting the overall responsiveness of the web

Measurement	Description	Measurement Unit	Interpretation
			site/application. This is why, if the <i>Average browser initial request time</i> measure reports an abnormally high value, you will have to compare the values of the <i>Average browser wait time</i>, <i>Average redirection time</i>, and <i>Average AppCache time</i> measure to determine whether/not the initial request delay observed on the browser is because requests have been waiting on the browser for too long a time. Some of the means by which you can reduce browser waits are briefly discussed below: • Browsers such as Mozilla Firefox support upto 10 parallel requests. You may want to recommend such browsers for your web site/web application users, so that more requests are processed and fewer requests are queued on the browser, thus reducing browser wait time. • Web site/application developers can try domain sharding - i.e., split content across multiple domains. Typically, when a user connects to a web page, his or her browser scans the resulting HTML for resources to download. Normally these resources are supplied by a single domain - the domain providing the web page or a domain created specifically for resources. With domain sharding, the user's browser

Measurement	Description	Measurement Unit	Interpretation
			connects to two or more different domains to simultaneously download the resources needed to render the web page. This allows website/application to be delivered faster to users as they do not have to wait for the previous set of requests to end before beginning the next set.
Average redirection time	Indicates the time that requests from this city spent in redirection before fetching the pages.	ms	In 3.6.6, this is the elapsed time between the redirectStart and redirectEnd events. URL redirection, also known as URL forwarding, is a technique to give a page, a form, or a whole web site/application, more than one URL address. Usually, web site administrators use URL redirection to: • Redirect users to the mobile version of the site • Redirect users to secured pages • Redirect users to the latest version of the resource/content • Redirect users to pages specific to their geo location • Redirecting canonical URLs Though redirects are useful, they have to be kept at a minimum, as each redirect on a page adds latency to the overall page load time. This is because, when a user enters a domain into the browser and hits enter, the DNS resolution process is triggered and the domain is resolved to its corresponding IP address in a few

Measurement	Description	Measurement Unit	Interpretation
			milliseconds. If the landing page has another redirect, then the browser repeats the entire DNS resolution process once again to guide the user to the correct web page. The multiple redirect requests are taxing on the browser resources and slow down the page load. Web page load time is also affected by internal redirects; for example, if the page tries to load content from a URL that has been redirected to newer or updated content, then the browser must create additional requests to fetch the valid content. These redirects result in additional round trips between the browser and the web server which pushes the load time higher; the perceived performance is degraded every time the browser encounters a redundant redirect. Web site/application performance is also impacted if redirects are not implemented correctly. Some of the common redirect errors are: • Multiple redirects: The higher the number of redirects on a page, the higher is its page load time. • Invalid redirects: There are often instances where the web site administrator sets up bulk redirects without verifying the validity of the redirects. The site may also have old redirects that were never cleaned up. This can cause several issues on the site like broken links and 404s.

Measurement	Description	Measurement Unit	Interpretation
			• Redirect loop: When there are several redirects on the page that are linked to each other, it creates a chain of redirects which may loop back to the same URL that initiated the redirect. This results in a redirect loop error and the user will not be able to access the site. Therefore, if you find that the value of the <i>Average page load time</i> measure is abnormally high owing to an unusually high value for the <i>Average redirection time</i> measure, then make sure you follow the best practices outlined below while implementing redirects, so you can significantly reduce page load time and improve user experience: • Avoid redundant redirects: It's recommended to avoid redirects where possible and to use this method only when absolutely needed. This will cut down unnecessary overhead and improve the perceived performance of the page. • Chain redirects: When a URL is linked to another URL, this creates a chained redirect. Each URL added to the chain adds latency to the page. Chained redirects have a negative impact not only on page speed, but also SEO. • Clean-up redirects: You may have hundreds of redirects on your website

Measurement	Description	Measurement Unit	Interpretation
			and it could be one of the main factors affecting page speed. Old redirects may conflict with new URLs, backlinks can cause odd errors on the page. It is recommended to verify all the redirects you have set up and to remove the ones that are no longer needed. Retain the old links that have major referral traffic, while those that are rarely accessed can be removed. This exercise will help improve page speed significantly.
Average AppCache time	Indicates the time taken to check whether/not the requests from this city can be serviced by the AppCache.	ms	HTML5 provides an application caching mechanism that lets web-based applications run offline. Developers can use the Application Cache (AppCache) interface to specify resources that the browser should cache and make available to offline users. Applications that are cached load and work correctly even if users click the refresh button when they are offline. Using an application cache gives an application the following benefits: • Offline browsing: users can navigate a site even when they are offline. • Speed: cached resources are local, and therefore load faster. • Reduced server load: the browser only downloads resources that have changed from the server. To enable the application cache for an application, you must include the manifest attribute in the <html> element in your application's pages. The manifest attribute references a cache manifest file, which is a text file that lists resources (files) that

Measurement	Description	Measurement Unit	Interpretation
			the browser should cache for your application. The browser does not cache pages that do not contain the manifest attribute, unless such pages are explicitly listed in the manifest file itself. You do not need to list all the pages you want cached in the manifest file, the browser implicitly adds every page that the user visits and that has the manifest attribute set to the application cache. When the browser visits a document that includes the manifest attribute, if no application cache exists, the browser loads the document and then fetches all the entries listed in the manifest file, creating the first version of the application cache. Subsequent visits to that document cause the browser to load the document and other assets specified in the manifest file from the application cache. If the manifest file has changed, all the files listed in the manifest—as well as those already added to the cache—are fetched into a temporary cache. Once all the files have been successfully retrieved, they are moved into the real offline cache automatically. Since the document has already been loaded into the browser from the cache, the updated document will not be rendered until the document is reloaded (either manually or programmatically). A high value of this measure signifies that requests are spending too much time in the AppCache. This also introduces page loading latencies, which have an adverse

Measurement	Description	Measurement Unit	Interpretation
			effect on user-perceived performance of a web site/application. Common reasons for AppCaching issues and their practical solutions are detailed below: • If the media type is not set, then AppCache will not work. To avoid this, make sure that the manifest file is always served under the correct media type of text/cache-manifest. • If the manifest file is not served to the web browser from the same origin as the host page, the manifest file will fail to load. To avoid this, make sure that the manifest file is always served from the same origin as the host page. However, note that the manifest file can hold reference to resources to be cached from other domains. • The relative URLs that you mention in the manifest are relative to the manifest file and not to the document where you reference the manifest file. If you make this error when the manifest and the reference are not in the same path, the resources will fail to load and in turn the manifest file will not be loaded. This will stall appcaching. • Any change made to the manifest file will cause the entire set of files to be downloaded again. Moreover, if a manifest file is added to a HTML file, it

Measurement	Description	Measurement Unit	Interpretation
			forces all resources to be downloaded synchronously as soon as the manifest file is downloaded. As a result, resources that may not yet be required, such as JavaScript or an image below the fold, will be downloaded at the start of the page. This can increase page load time significantly. The solution to this is to load the Application Cache from a simple HTML file loaded in an iframe. This not only avoids caching dynamic HTML but also avoids the Application Cache being downloaded asynchronously after the page load has completed.
Average DNS lookup time	Indicates the time taken by the browser to perform the domain lookup for connecting to the pages requested by this city.	ms	A high value for this measure will not only affect DNS lookup, but will also impact the <i>Average network time</i> and <i>Average page load time</i> 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 for accessing pages from this city.	ms	A bad network connection between the browser client and the server can delay TCP connections to the server. As a result, the <i>Average network time</i> too will increase, thus impacting page load time and overall user experience with the web site/web application.
Average SSL handshake time	Indicates the time taken by requests from this city to complete the SSL handshake.	ms	An SSL handshake happens when a browser makes a secure request for content, also known as an encrypted HTTPS connection. The user's browser and server negotiate encrypted keys and certificates to establish a secure connection between each other. Because this SSL negotiation requires exchanges between the browser and your server, it increases the time spent by the request on

Measurement	Description	Measurement Unit	Interpretation
			the network (i.e., it adds to the value of the Average network time measure). This in turn increases page load time. In fact, an SSL handshake, along with DNS lookup and TCP handshake add three round trips to the page load time. A quick fix to this is to use HTTP/2. HTTP/2 can use caching to reduce SSL setup to only one round trip.

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 3.206 below provides a sneak-peek at the standard columns that are available as part of the detailed diagnosis of all measures.

USER EXPERIENCE	REQUEST DATE	USER NAME	URL	PAGETYPE	PAGE LOAD TIME (MS)	BROWSER INITIAL REQUEST TIME (MS)	NETWORK TIME (MS)	SERVEF
 Slow 	Dec 24, 2019 00:56:07 EDT	Mary	/EasyKart/ShippingPage.jsp	Page	29851	0	32	25413
 Slow 	Dec 23, 2019 01:36:09 EDT	Jennifer	/EasyKart/ShippingPage.jsp	Page	29851	0	32	25413
 Slow 	Dec 23, 2019 01:26:08 EDT	Brian	/EasyKart/ShippingPage.jsp	Page	29851	0	32	25413
 Slow 	Dec 23, 2019 01:06:06 EDT	Matthew	/EasyKart/ShippingPage.jsp	Page	29851	0	32	25413
 Slow 	Dec 24, 2019 00:56:06 EDT	Amanda	/EasyKart/ShippingPage.jsp	Page	29851	0	32	25413
 Slow 	Dec 23, 2019 01:11:08 EDT	Donald	/EasyKart/AddToCart.jsp	Page	29459	0	32	25077
 Slow 	Dec 23, 2019 01:11:08 EDT	Susan	/EasyKart/AddToCart.jsp	Page	29459	0	32	25077
 Slow 	Dec 23, 2019 01:36:09 EDT	Matthew	/EasyKart/BrowseProducts.jsp	Page	29400	0	32	25026
 Slow 	Dec 24, 2019 00:51:07 EDT	Deborah	/EasyKart/PaymentPage.jsp	Page	29217	0	32	24869
 Slow 	Dec 23, 2019 01:31:08 EDT	Anthony	/EasyKart/PaymentPage.jsp	Page	29217	0	32	24869

Figure 3.206: 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;
- Which user is accessing each page? **Note that the user name will be captured and displayed only if:**
 - The 'Capture UserName' flag in the Add/Modify page of the Real User Monitor component is set to 'Yes' (see Figure 2), and;
 - The procedure detailed in the Enabling the eG Real User Monitor to Capture User Names topic is followed.
- 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? or page rendering?
- Which browser and device was used to access each page?
- Which is the device OS?
- From which client was each page request received?
- From which country, city, and region was each page accessed?

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 for each measurement period in the timeline chosen for the 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 per measurement period in your detailed diagnosis.

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

Page views

The following will be displayed for each measurement period in the chosen timeline:

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

USER EXPERIENCE	REQUEST DATE	USER NAME	URL	PAGETYPE	PAGE LOAD TIME (MS)	BROWSER INITIAL REQUEST TIME (MS)	NETWORK TIME (MS)	SERVEF
Slow	Dec 23, 2019 01:26:08 EDT	Brian	/EasyKart/ShippingPage.jsp	Page	29851	0	32	25413
Slow	Dec 23, 2019 01:36:09 EDT	Jennifer	/EasyKart/ShippingPage.jsp	Page	29851	0	32	25413
Slow	Dec 23, 2019 01:36:09 EDT	Matthew	/EasyKart/BrowseProducts.jsp	Page	29400	0	32	25026
Slow	Dec 23, 2019 01:31:08 EDT	Anthony	/EasyKart/PaymentPage.jsp	Page	29217	0	32	24869
Slow	Dec 23, 2019 01:46:06 EDT	Karen	/EasyKart/AddToCart.jsp	Page	29105	0	32	24773
Slow	Dec 23, 2019 02:11:06 EDT	Joseph	/EasyKart/AddToCart.jsp	Page	29105	0	32	24773
Slow	Dec 23, 2019 01:26:08 EDT	Brian	/EasyKart/PaymentPage.jsp	Page	28912	0	32	24607
Slow	Dec 23, 2019 01:36:09 EDT	Jennifer	/EasyKart/PaymentPage.jsp	Page	28912	0	32	24607
Slow	Dec 23, 2019 02:01:09 EDT	Jennifer	/EasyKart/AddToCart.jsp	Page	28873	0	32	24574
Slow	Dec 23, 2019 02:01:09 EDT	David	/EasyKart/AddToCart.jsp	Page	28873	0	32	24574

Figure 3.207: The detailed diagnosis of the Page views measure

Average page load time

The following will be displayed for each measurement period in the chosen timeline:

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

USER EXPERIENCE	REQUEST DATE	USER NAME	URL	PAGETYPE	PAGE LOAD TIME (MS)	BROWSER INITIAL REQUEST TIME (MS)	NETWORK TIME (MS)	SERVEF
Slow	Dec 24, 2019 00:56:07 EDT	Mary	/EasyKart/ShippingPage.jsp	Page	29851	0	32	25413
Slow	Dec 23, 2019 01:36:09 EDT	Jennifer	/EasyKart/ShippingPage.jsp	Page	29851	0	32	25413
Slow	Dec 23, 2019 01:26:08 EDT	Brian	/EasyKart/ShippingPage.jsp	Page	29851	0	32	25413
Slow	Dec 23, 2019 01:06:06 EDT	Matthew	/EasyKart/ShippingPage.jsp	Page	29851	0	32	25413
Slow	Dec 24, 2019 00:56:06 EDT	Amanda	/EasyKart/ShippingPage.jsp	Page	29851	0	32	25413
Slow	Dec 23, 2019 01:11:08 EDT	Donald	/EasyKart/AddToCart.jsp	Page	29459	0	32	25077
Slow	Dec 23, 2019 01:11:08 EDT	Susan	/EasyKart/AddToCart.jsp	Page	29459	0	32	25077
Slow	Dec 23, 2019 01:36:09 EDT	Matthew	/EasyKart/BrowseProducts.jsp	Page	29400	0	32	25026
Slow	Dec 24, 2019 00:51:07 EDT	Deborah	/EasyKart/PaymentPage.jsp	Page	29217	0	32	24869
Slow	Dec 23, 2019 01:31:08 EDT	Anthony	/EasyKart/PaymentPage.jsp	Page	29217	0	32	24869

Figure 3.208: The detailed diagnosis of the Average page load time measure

JavaScript error page views and JavaScript error page view percentage

The following will be displayed for each measurement period in the chosen timeline:

- The top-5 most recent page views that encountered JavaScript errors;

USER EXPERIENCE	REQUEST DATE	USER NAME	URL	PAGETYPE	PAGE LOAD TIME (MS)	BROWSER INITIAL REQUEST TIME (MS)	NETWORK TIME (MS)
Error	Dec 23, 2019 01:41:09 EDT	Mark	/corebanking/retail/feesandcharges.jsp	Page	4943	7	5
Error	Dec 23, 2019 02:21:06 EDT	Steven	/corebanking/retail/feesandcharges.jsp	Page	4943	7	5
Error	Dec 23, 2019 01:46:09 EDT	Christop	/corebanking/retail/feesandcharges.jsp	Page	4943	7	5
Error	Dec 23, 2019 01:31:07 EDT	Kenneth	/corebanking/retail/chequestatusenquiry.jsp	Page	4652	7	5
Error	Dec 23, 2019 02:06:08 EDT	Matthew	/corebanking/retail/chequestatusenquiry.jsp	Page	4652	7	5
Error	Dec 23, 2019 01:51:06 EDT	Emily	/corebanking/retail/chequestatusenquiry.jsp	Page	4652	7	5
Error	Dec 23, 2019 01:31:07 EDT	Kenneth	/corebanking/retail/chequestatusenquiry.jsp	Page	4374	7	5
Error	Dec 23, 2019 01:51:06 EDT	Emily	/corebanking/retail/chequestatusenquiry.jsp	Page	4374	7	5
Error	Dec 23, 2019 01:46:09 EDT	Christop	/corebanking/retail/chequestatusenquiry.jsp	Page	4374	7	5
Error	Dec 23, 2019 01:41:09 EDT	Mark	/corebanking/retail/chequestatusenquiry.jsp	Page	4374	7	5

Figure 3.209: The detailed diagnosis of the JavaScript errors measure

Slow page views

The following will be displayed for each measurement period in the chosen timeline:

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

USER EXPERIENCE	REQUEST DATE	USER NAME	URL	PAGETYPE	PAGE LOAD TIME (MS)	BROWSER INITIAL REQUEST TIME (MS)	NETWORK TIME (MS)
Slow	Dec 23, 2019 01:31:07 EDT	Kenneth	/corebanking/retail/feesandcharges.jsp	Page	4943	7	5
Slow	Dec 23, 2019 02:06:08 EDT	Matthew	/corebanking/retail/feesandcharges.jsp	Page	4943	7	5
Slow	Dec 23, 2019 01:51:06 EDT	Emily	/corebanking/retail/feesandcharges.jsp	Page	4943	7	5
Slow	Dec 23, 2019 01:31:07 EDT	Kenneth	/corebanking/retail/feesandcharges.jsp	Page	4900	7	5
Slow	Dec 23, 2019 02:21:06 EDT	Steven	/corebanking/retail/feesandcharges.jsp	Page	4900	7	5
Slow	Dec 23, 2019 02:06:08 EDT	Matthew	/corebanking/retail/feesandcharges.jsp	Page	4900	7	5
Slow	Dec 23, 2019 01:51:06 EDT	Emily	/corebanking/retail/feesandcharges.jsp	Page	4900	7	5
Slow	Dec 23, 2019 01:46:09 EDT	Christop	/corebanking/retail/feesandcharges.jsp	Page	4900	7	5
Slow	Dec 23, 2019 01:41:09 EDT	Mark	/corebanking/retail/feesandcharges.jsp	Page	4900	7	5
Slow	Dec 23, 2019 01:36:08 EDT	Emily	/corebanking/retail/chequestatusenquiry.jsp	Page	4652	7	5

Figure 3.210: The detailed diagnosis of the Slow page views measure

Tolerating page views

The following will be displayed for each measurement period in the chosen timeline:

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

USER EXPERIENCE	REQUEST DATE	USER NAME	URL	PAGETYPE	PAGE LOAD TIME (MS)	BROWSER INITIAL REQUEST TIME (MS)	NETWORK TIME (MS)
Slow	Dec 23, 2019 01:31:07 EDT	Kenneth	/corebanking/retail/standinginstruction.jsp	Page	3871	7	5
Slow	Dec 23, 2019 02:26:07 EDT	Karen	/corebanking/retail/standinginstruction.jsp	Page	3871	7	5
Slow	Dec 23, 2019 02:16:07 EDT	John	/corebanking/retail/standinginstruction.jsp	Page	3871	7	5
Slow	Dec 23, 2019 02:11:07 EDT	Sharon	/corebanking/retail/standinginstruction.jsp	Page	3871	7	5
Slow	Dec 23, 2019 02:06:08 EDT	Matthew	/corebanking/retail/standinginstruction.jsp	Page	3871	7	5
Slow	Dec 23, 2019 02:01:06 EDT	Dorothy	/corebanking/retail/standinginstruction.jsp	Page	3871	7	5
Slow	Dec 23, 2019 01:56:08 EDT	Sarah	/corebanking/retail/standinginstruction.jsp	Page	3871	7	5
Slow	Dec 23, 2019 01:51:06 EDT	Emily	/corebanking/retail/standinginstruction.jsp	Page	3871	7	5
Slow	Dec 23, 2019 01:36:08 EDT	Emily	/corebanking/retail/standinginstruction.jsp	Page	3871	7	5
Slow	Dec 23, 2019 01:36:08 EDT	Emily	/corebanking/retail/branchlocator.jsp	Page	3709	7	4

Figure 3.211: The detailed diagnosis of the Tolerating page views measure

Frustrated page views

The following will be displayed for each measurement period in the chosen timeline:

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

USER EXPERIENCE	REQUEST DATE	USER NAME	URL	PAGETYPE	PAGE LOAD TIME (MS)	BROWSER INITIAL REQUEST TIME (MS)	NETWORK TIME (MS)
 Slow	Dec 23, 2019 01:31:07 EDT	Kenneth	/corebanking/retail/feesandcharges.jsp	Page	4943	7	5
 Error	Dec 23, 2019 02:21:06 EDT	Steven	/corebanking/retail/feesandcharges.jsp	Page	4943	7	5
 Slow	Dec 23, 2019 02:06:08 EDT	Matthew	/corebanking/retail/feesandcharges.jsp	Page	4943	7	5
 Slow	Dec 23, 2019 01:51:06 EDT	Emily	/corebanking/retail/feesandcharges.jsp	Page	4943	7	5
 Error	Dec 23, 2019 01:46:09 EDT	Christop	/corebanking/retail/feesandcharges.jsp	Page	4943	7	5
 Error	Dec 23, 2019 01:41:09 EDT	Mark	/corebanking/retail/feesandcharges.jsp	Page	4943	7	5
 Slow	Dec 23, 2019 01:31:07 EDT	Kenneth	/corebanking/retail/feesandcharges.jsp	Page	4900	7	5
 Slow	Dec 23, 2019 01:51:06 EDT	Emily	/corebanking/retail/feesandcharges.jsp	Page	4900	7	5
 Slow	Dec 23, 2019 01:46:09 EDT	Christop	/corebanking/retail/feesandcharges.jsp	Page	4900	7	5
 Slow	Dec 23, 2019 01:41:09 EDT	Mark	/corebanking/retail/feesandcharges.jsp	Page	4900	7	5

Figure 3.212: The detailed diagnosis of the Frustrated page views measure

Satisfied page views

The following will be displayed for each measurement period in the chosen timeline:

- The top-5 page views (in terms of time of request), 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 Sitetest**. **Note that such page views can have JavaScript errors.**

USER EXPERIENCE	REQUEST DATE	USER NAME	URL	PAGETYPE	PAGE LOAD TIME (MS)	BROWSER INITIAL REQUEST TIME (MS)
 Healthy	Dec 31, 2019 15:43:58 IST	-	/contact	Page	1993	303
 Healthy	Dec 31, 2019 15:43:57 IST	-	/sockjs-node/info	Ajax	17	N/A
 Healthy	Dec 31, 2019 15:42:55 IST	-	/sockjs-node/info	Ajax	41	N/A
 Healthy	Dec 31, 2019 15:42:55 IST	-	/contact	Page	1653	303
 Healthy	Dec 31, 2019 15:41:53 IST	-	/contact	Page	1584	304
 Healthy	Dec 31, 2019 15:41:53 IST	-	/sockjs-node/info	Ajax	12	N/A
 Healthy	Dec 31, 2019 15:40:52 IST	-	/contact	Page	2381	304
 Healthy	Dec 31, 2019 15:40:51 IST	-	/sockjs-node/info	Ajax	9	N/A
 Healthy	Dec 31, 2019 15:39:50 IST	-	/contact	Page	1991	304
 Healthy	Dec 31, 2019 15:39:49 IST	-	/sockjs-node/info	Ajax	20	N/A

Figure 3.213: The detailed diagnosis of the Satisfied page views measure

Desktop page views

The following will be displayed for each measurement period in the chosen timeline:

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

Details of Page Requests from Desktops							
USER EXPERIENCE	REQUEST DATE	USER NAME	URL	PAGETYPE	PAGE LOAD TIME (MS)	BROWSER INITIAL REQUEST TIME (MS)	NETWORK TIME (MS)
 Slow 	Dec 23, 2019 03:11:07 EDT	Christop	/corebanking/retail/feesandcharges.jsp	Page	4943	7	5
 Slow 	Dec 23, 2019 03:11:07 EDT	Christop	/corebanking/retail/feesandcharges.jsp	Page	4900	7	5
 Slow 	Dec 23, 2019 02:56:06 EDT	Mary	/corebanking/retail/chequestatusenquiry.jsp	Page	4652	7	5
 Slow 	Dec 23, 2019 03:41:06 EDT	Ashley	/corebanking/retail/chequestatusenquiry.jsp	Page	4652	7	5
 Slow 	Dec 23, 2019 03:21:09 EDT	Edward	/corebanking/retail/chequestatusenquiry.jsp	Page	4652	7	5
 Slow 	Dec 23, 2019 03:16:07 EDT	Deborah	/corebanking/retail/chequestatusenquiry.jsp	Page	4652	7	5
 Error 	Dec 23, 2019 03:11:07 EDT	Christop	/corebanking/retail/chequestatusenquiry.jsp	Page	4652	7	5
 Slow 	Dec 23, 2019 02:56:06 EDT	Mary	/corebanking/retail/chequestatusenquiry.jsp	Page	4374	7	5
 Slow 	Dec 23, 2019 03:16:07 EDT	Deborah	/corebanking/retail/chequestatusenquiry.jsp	Page	4374	7	5
 Error 	Dec 23, 2019 03:11:07 EDT	Christop	/corebanking/retail/chequestatusenquiry.jsp	Page	4374	7	5

Figure 3.214: The detailed diagnosis of the Desktop page views measure

Mobile page views

The following will be displayed for each measurement period in the chosen timeline:

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

Details of Page Requests from Mobiles							
USER EXPERIENCE	REQUEST DATE	USER NAME	URL	PAGETYPE	PAGE LOAD TIME (MS)	BROWSER INITIAL REQUEST TIME (MS)	NETWORK TIME (MS)
 Slow 	Dec 23, 2019 03:01:08 EDT	Margaret	/corebanking/retail/chequestatusenquiry.jsp	Page	4652	7	5
 Slow 	Dec 23, 2019 03:31:09 EDT	Rebecca	/corebanking/retail/chequestatusenquiry.jsp	Page	4652	7	5
 Slow 	Dec 23, 2019 03:06:08 EDT	Donna	/corebanking/retail/chequestatusenquiry.jsp	Page	4652	7	5
 Slow 	Dec 23, 2019 03:01:08 EDT	Margaret	/corebanking/retail/chequestatusenquiry.jsp	Page	4374	7	5
 Slow 	Dec 23, 2019 03:31:09 EDT	Rebecca	/corebanking/retail/chequestatusenquiry.jsp	Page	4374	7	5
 Slow 	Dec 23, 2019 03:06:08 EDT	Donna	/corebanking/retail/chequestatusenquiry.jsp	Page	4374	7	5
 Slow 	Dec 23, 2019 03:01:08 EDT	Margaret	/corebanking/retail/standinginstruction.jsp	Page	3871	7	5
 Slow 	Dec 23, 2019 03:31:09 EDT	Rebecca	/corebanking/retail/standinginstruction.jsp	Page	3871	7	5
 Slow 	Dec 23, 2019 03:06:08 EDT	Donna	/corebanking/retail/standinginstruction.jsp	Page	3871	7	5
 Slow 	Dec 23, 2019 03:01:08 EDT	Margaret	/corebanking/retail/branchlocator.jsp	Page	3709	7	4

Figure 3.215: The detailed diagnosis of the Mobile page views measure

Tablet page views

The following will be displayed for each measurement period in the chosen timeline:

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

USER EXPERIENCE	REQUEST DATE	USER NAME	URL	PAGETYPE	PAGE LOAD TIME (MS)	BROWSER INITIAL REQUEST TIME (MS)	NETWORK TIME (MS)
 Slow 	Dec 23, 2019 03:26:09 EDT	Ryan	/corebanking/retail/chequestatusenquiry.jsp	Page	4652	7	5
 Slow 	Dec 23, 2019 03:51:06 EDT	Dorothy	/corebanking/retail/chequestatusenquiry.jsp	Page	4652	7	5
 Slow 	Dec 23, 2019 03:46:08 EDT	Andrew	/corebanking/retail/chequestatusenquiry.jsp	Page	4652	7	5
 Slow 	Dec 23, 2019 03:36:08 EDT	Jennifer	/corebanking/retail/chequestatusenquiry.jsp	Page	4652	7	5
 Slow 	Dec 23, 2019 03:26:09 EDT	Ryan	/corebanking/retail/chequestatusenquiry.jsp	Page	4374	7	5
 Slow 	Dec 23, 2019 03:51:06 EDT	Dorothy	/corebanking/retail/chequestatusenquiry.jsp	Page	4374	7	5
 Slow 	Dec 23, 2019 03:46:08 EDT	Andrew	/corebanking/retail/chequestatusenquiry.jsp	Page	4374	7	5
 Slow 	Dec 23, 2019 03:36:08 EDT	Jennifer	/corebanking/retail/chequestatusenquiry.jsp	Page	4374	7	5
 Slow 	Dec 23, 2019 03:26:09 EDT	Ryan	/corebanking/retail/standinginstruction.jsp	Page	3871	7	5
 Slow 	Dec 23, 2019 03:36:08 EDT	Jennifer	/corebanking/retail/standinginstruction.jsp	Page	3871	7	5

Figure 3.216: 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 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 the magnifying glass icon 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 determine where your request was delayed. If the pie chart reveals that *Browser* time is the reason for the poor responsiveness of a URL, then a quick look at the *Browser Request* and *Browser Response* break-up in the transaction flow chart will point you to why the request lost time on the browser - is it because the browser was slow in connecting to the server and processing the request? or is it because the browser was slow in fetching the resources from the server and rendering the requested page? If the issue was with the *Browser Request*, then the transaction flow chart also reveals why initial request processing by the browser was delayed - is it because the URL took too much time to follow redirection? or is it because the URL was waiting too long for a previous request to end? or is an AppCaching issue the reason?

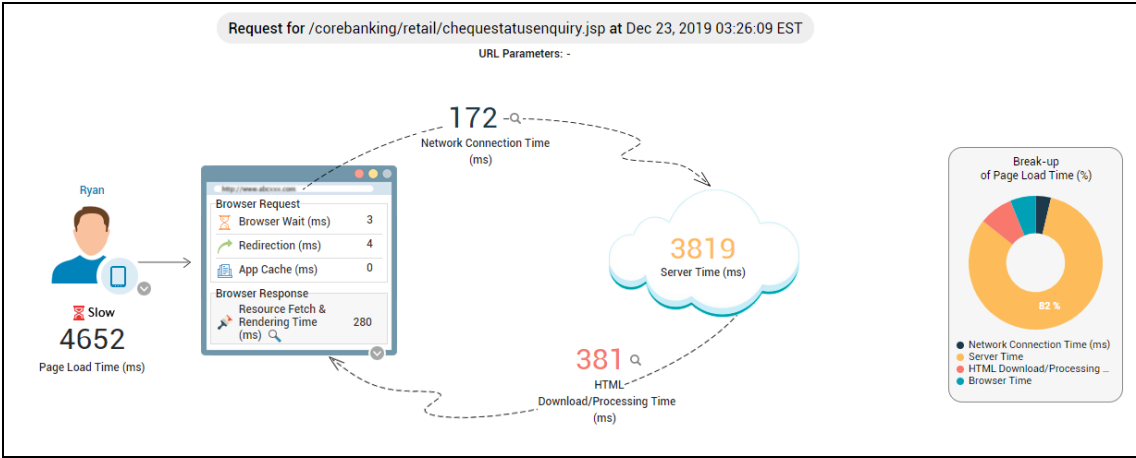


Figure 3.217: The RUM Transaction Details page displaying the entire flow of the page request/transaction

On the other hand, if *Browser Response* is the bottleneck, then you click on the 'magnifying glass' icon under *Browser Response* in Figure 3.218 to know why. The **Resource Details** tab page will then open listing all the resources the browser was attempting to download from the server in response to the URL request, and the time taken to download each resource. This list enables administrators to pinpoint the precise resources that took an abnormally long time to be downloaded, thereby delaying page rendering. For instance, if resources such as CSS or JavaScripts are taking too long to load, they can be identified easily using Figure 3.218. This way, web masters can get insight into what resources to minify to get faster page load. Also, if there is a scenario where resources are being downloaded during each page load instead of being served from the browser cache, it can be found using Figure 3.218.

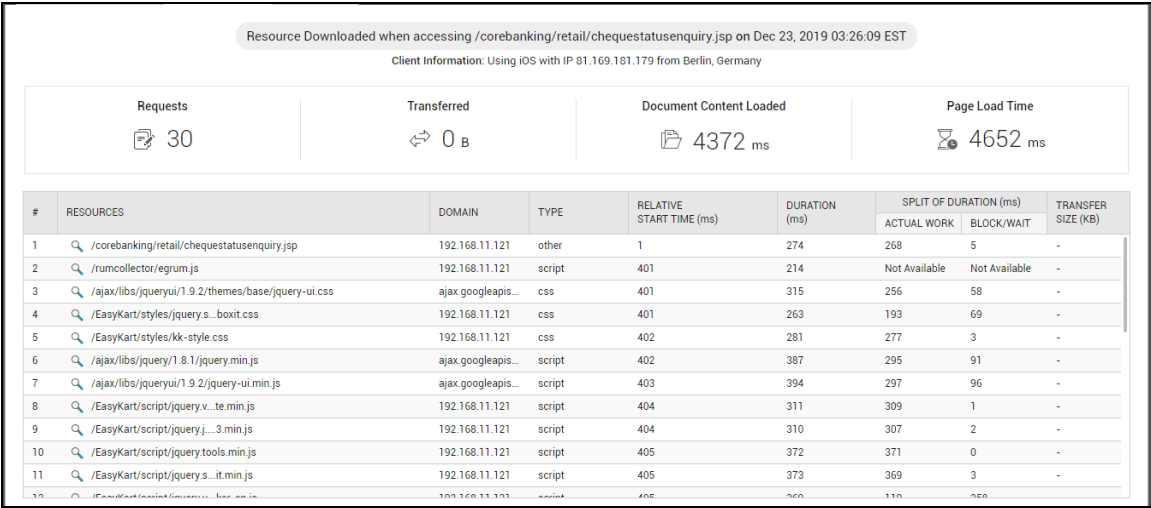


Figure 3.218: The Resource Details tab page revealing the resources that took too long to be downloaded from the server

To know where resource download was bottlenecked, click on the 'magnifying glass' icon alongside the problematic resource in Figure 3.218. Figure 3.219 will then appear.

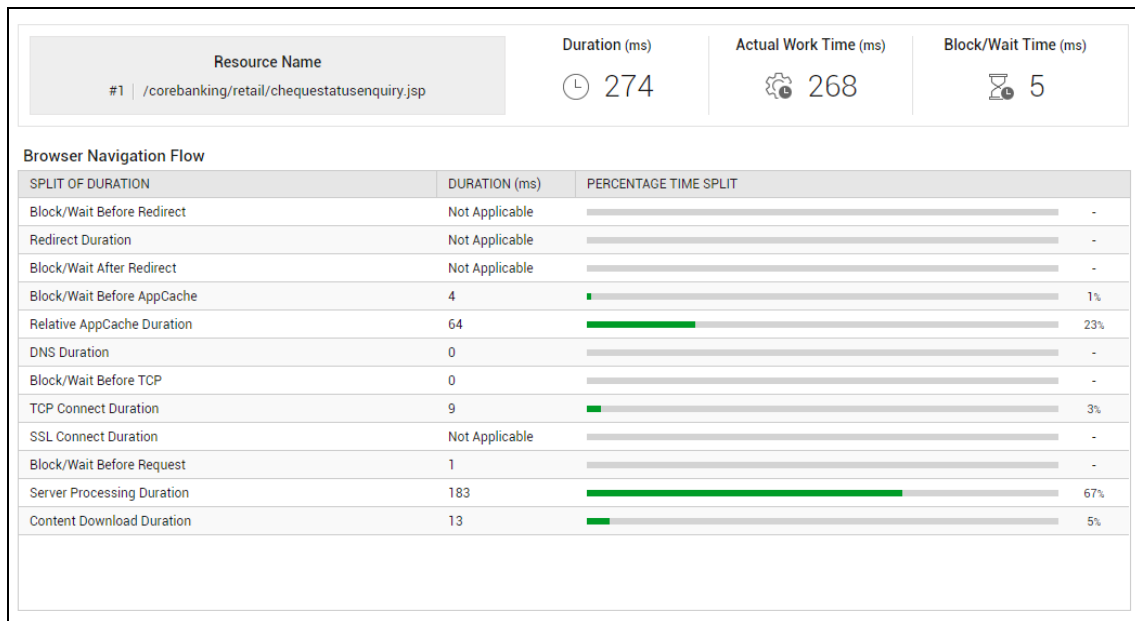


Figure 3.219: Resource fetch time break-up

For a chosen resource, Figure 3.219 quickly reveals where resource download was delayed - on the browser? the network? or the server? If the delay was at the browser or the network, then the granular insights provided by Figure 3.219 will enable web masters to accurately identify the exact browser/network activity that caused resource loading slowness - redirection? AppCaching? browser wait? TCP connection? DNS lookup? or SSL handshake?

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 in the SSL handshake?

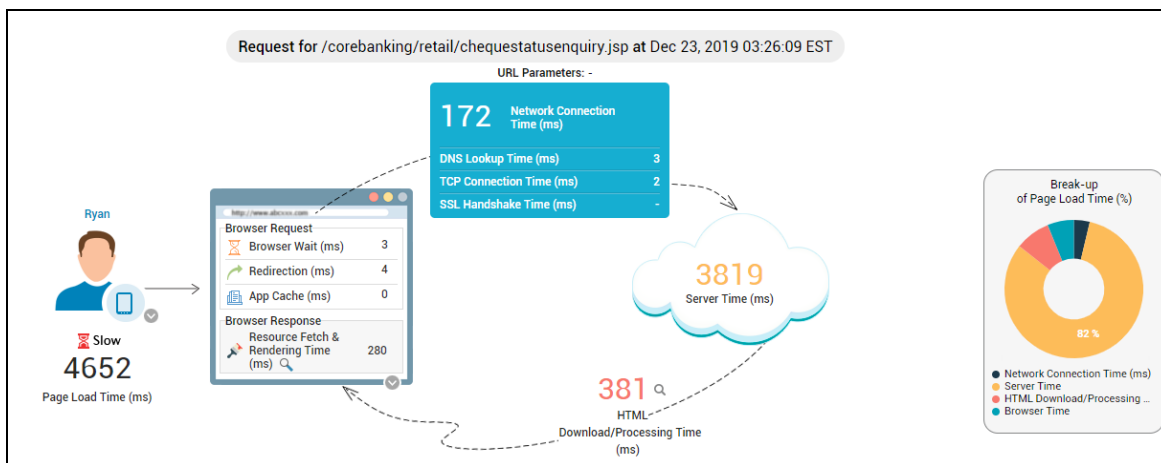


Figure 3.220: The transaction flow indicating the reason why Network connection time is high

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?

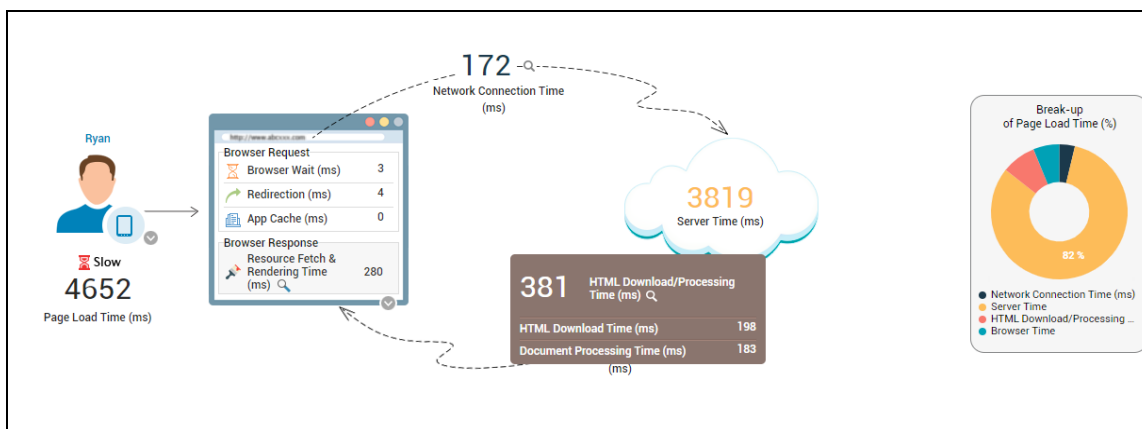


Figure 3.221: The transaction flow chart displaying the break-up of Content Download Time

If a web page takes too long to download content, then users can quickly drill down from the detailed diagnostics of the content download time measure, shown by Figure 3.221, to view the load time of each resource in that page (see Figure 3.218). For this, click on the 'magnifying glass' icon alongside **HTML Download/Processing Time (ms)** in Figure 3.221. This enables administrators to identify the exact resource in that page that is delaying the content download. You can even drill down from a particular resource in Figure 3.218 to view where resource loading was bottlenecked (see Figure 3.219).

Note:

eG Enterprise provides the resource-level visibility into page load time (see Figure 3.218 and Figure 3.219), by leveraging the Resource Timing API. The Resource Timing API enables retrieving and analyzing detailed network timing data regarding the loading of a web application's resource(s). A resource can be an XMLHttpRequest, image, script, etc. Now, if the users are accessing the monitored web page using the Internet Explorer (IE) browser, then sometimes, the load time of one/more resources may be displayed as -1 in the eG monitoring console. This is because, when monitoring page views from IE, the Resource Timing API reports '-1' as the load time of certain resources. Also, some versions of Internet Explorer, do not support the Resource Timing API at all. When monitoring page views from such versions of IE, no resource metrics will be reported.

The transaction flow diagram also provides you with useful information about the page being viewed such as the browser used to access the page, the page type, and redirection URL (if any). To view this information, click on the 'down arrow' icon you will find at the right, bottom corner of the Browser Request window, as depicted by Figure 3.222.

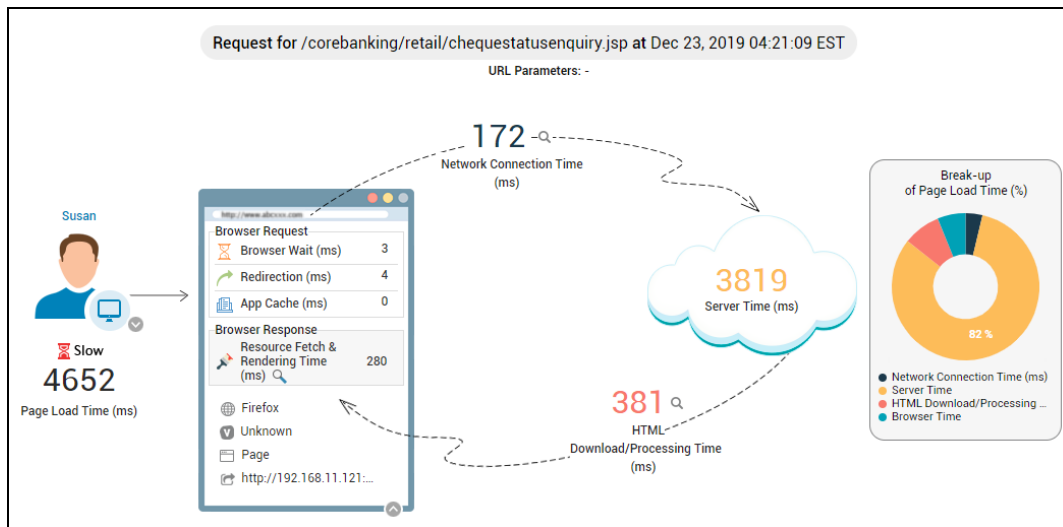


Figure 3.222: The transaction flow displaying page information

You can also view user-related information in the transaction flow diagram. The geography to which the user who initiated the transaction belongs, client from which he/she is connecting, and the operating system of the client can be viewed by simply clicking the 'down arrow' icon adjacent to the user icon, as depicted by Figure 3.223.

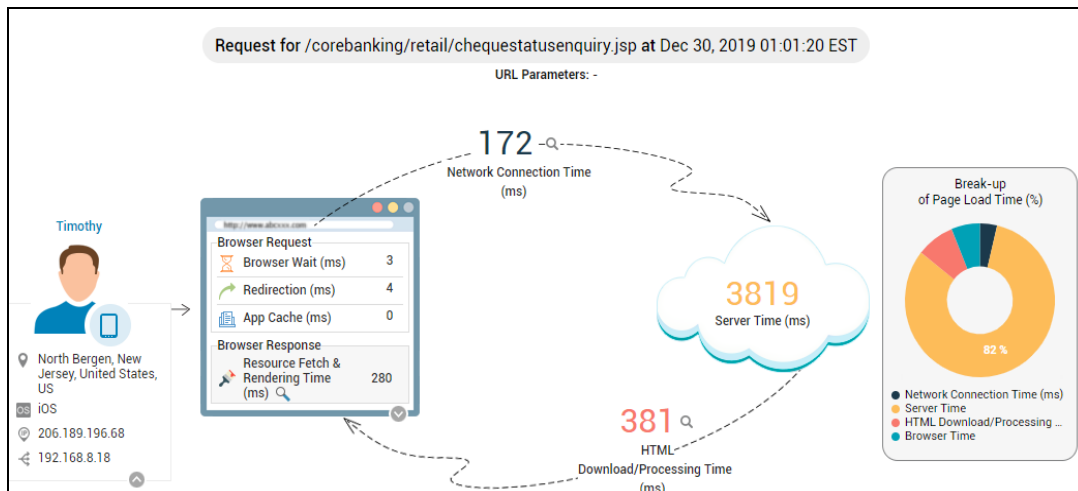


Figure 3.223: The transaction flow displaying user information

Ajax requests are depicted differently in the **RUM Transaction Details** page. If a request makes an Ajax call to the server, then such a call will be tagged as an **Ajax Call** in this page (see Figure 3.224).

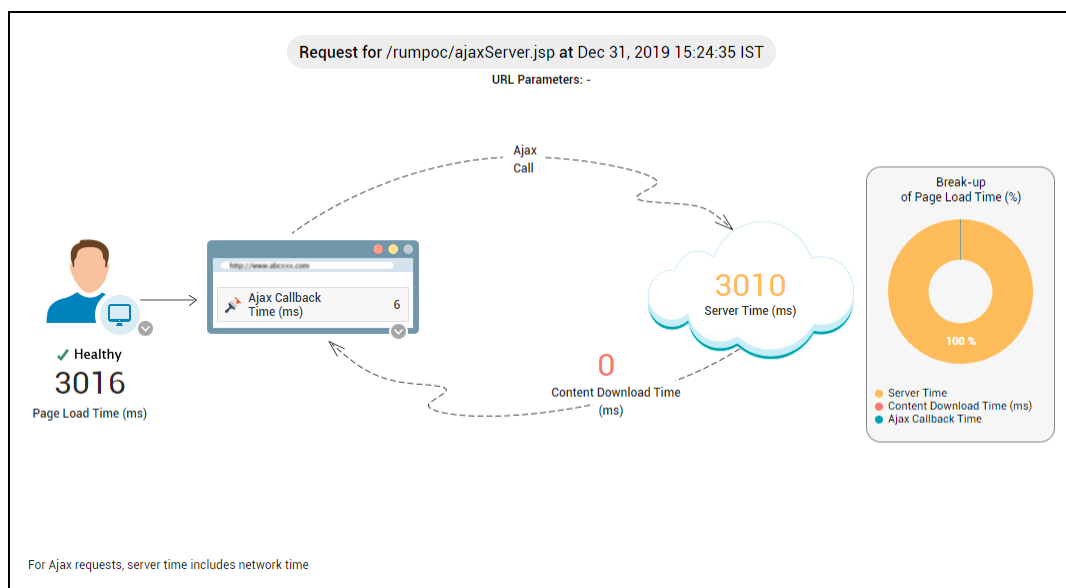


Figure 3.224: The RUM Transaction Details for an Ajax call

For Ajax requests, the **Network connection time** is included as part of **Server time**. This is why, in the **RUM Transaction Details** page for an Ajax request, the **Network connection time is not indicated** (see Figure 3.224). In the Detailed Diagnosis page also, the **Network Connection Time column** for an Ajax request will not display any value.

Typically, an Ajax function takes Ajax details as input along with a Callback reference. The Callback reference comes into play after the server processes the Ajax request. The Callback is generally used to communicate the result of the Ajax call to the caller, so that the caller can resume processing. For instance, say you have a function F1 which calls the Ajax function F2. F1 would like to know the result of the Ajax function F2 to proceed with its processing. For this, F1 will pass another function say C1 as an additional parameter to F2; C1 is the Callback reference. F2 will call C1 after it processes the Ajax request completely. C1 will communicate the result of F2 to the caller. The time taken by C1 - i.e., by the Callback function - for computing and sending the result to the caller is displayed in the **RUM Transaction Details** page as **Ajax Callback Time**.

Note:

- In the detailed diagnosis of an Ajax request, the **Ajax Callback Time** will be included in the **Browser Render Time** of that request.
- When reporting real-time metrics for an Ajax request, the **Ajax Callback Time** will be reported as **Document Processing Time**.

If an Ajax request is slow, then a quick look at Figure 3.224 will reveal to you what is delaying the request - server-side processing? Ajax Callback? or content download?

Now, if you click on an Error request in the detailed diagnosis, you will not only be able to view the path of that request in the **RUM Transaction Details** page, but you will also be able to view the precise error(s) that the page in question encountered, in the **Error Details tab** page (see Figure 3.225).

#	ERROR MESSAGE	ERROR STACK	ERROR METADATA	ERROR FILENAME	ERROR LINENUMBER	ERROR COL
1	Uncaught Error: error page7	-	-	http://localhost:42...	52	
2	Uncaught Error: error page2	-	-	http://localhost:42...	45	
3	Uncaught Error: error page3	-	-	http://localhost:42...	55	
4	Uncaught Error: error page6	-	-	http://localhost:42...	50	

Figure 3.225: Errors displayed in the Error Details tab page

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 3.226).

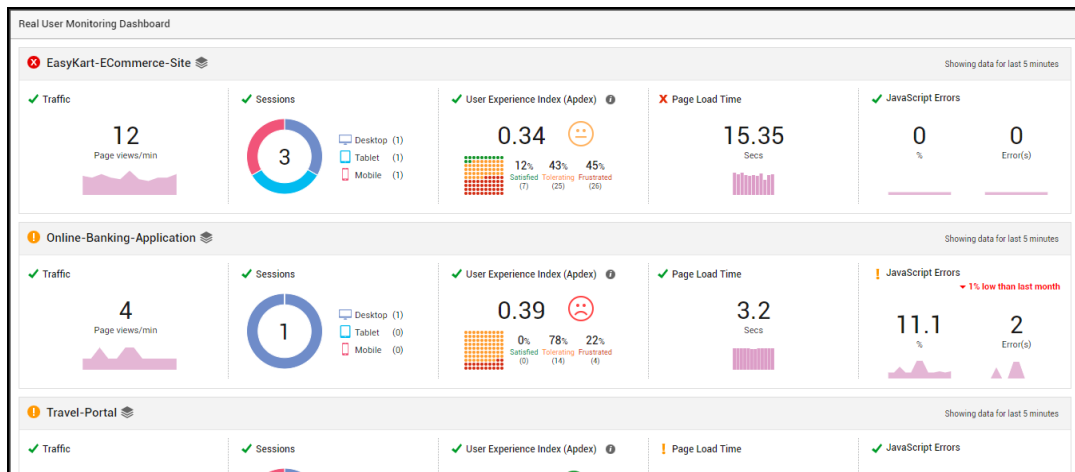


Figure 3.226: The Real User Monitor Dashboard

To access this dashboard, click on the *Real User Monitoring* option in the **Dashboards** tile of the **Monitor** menu.

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? Which is the most popular device
- 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 distinct users are currently connected to each web site / web application being monitored? - this is a good indicator of the number of users who are truly contributing to the traffic on the web site.

Note:

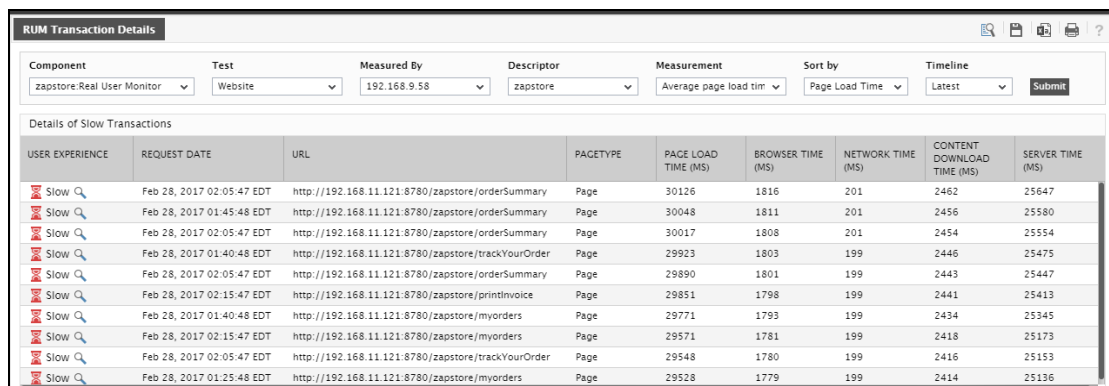
HttpOnly is a flag that can be set on cookies to prevent their contents from being accessed by JavaScript. This is often done for session cookies to hide the session identifier, as a security measure.

If the *HttpOnly* flag is set for the session cookies of the monitored web site, then the eG RUM JavaScript will not be able to read the session identifier from the cookies. In such cases therefore, the eG agent will not be able to report the count of unique sessions - i.e., the count of

distinct users currently connected to the web site. The **Unique Sessions** count will hence not be displayed in the RUM dashboard for of such web sites.

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 in the target application 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 3.227)



RUM Transaction Details									
Component	Test	Measured By	Descriptor	Measurement	Sort by	Timeline			
zapstore:Real User Monitor	Website	192.168.9.58	zapstore	Average page load tim	Page Load Time	Latest	Submit		
Details of Slow Transactions									
USER EXPERIENCE	REQUEST DATE	URL	PAGETYPE	PAGE LOAD TIME (MS)	BROWSER TIME (MS)	NETWORK TIME (MS)	CONTENT DOWNLOAD TIME (MS)	SERVER TIME (MS)	
Slow	Feb 28, 2017 02:05:47 EDT	http://192.168.11.121:8780/zapstore/orderSummary	Page	30126	1816	201	2462	25647	
Slow	Feb 28, 2017 01:45:48 EDT	http://192.168.11.121:8780/zapstore/orderSummary	Page	30048	1811	201	2456	25580	
Slow	Feb 28, 2017 02:05:47 EDT	http://192.168.11.121:8780/zapstore/orderSummary	Page	30017	1808	201	2454	25554	
Slow	Feb 28, 2017 01:40:48 EDT	http://192.168.11.121:8780/zapstore/trackYourOrder	Page	29923	1803	199	2446	25475	
Slow	Feb 28, 2017 02:05:47 EDT	http://192.168.11.121:8780/zapstore/orderSummary	Page	29890	1801	199	2443	25447	
Slow	Feb 28, 2017 02:15:47 EDT	http://192.168.11.121:8780/zapstore/printInvoice	Page	29851	1798	199	2441	25413	
Slow	Feb 28, 2017 01:40:48 EDT	http://192.168.11.121:8780/zapstore/myorders	Page	29771	1793	199	2434	25345	
Slow	Feb 28, 2017 02:15:47 EDT	http://192.168.11.121:8780/zapstore/myorders	Page	29571	1781	199	2418	25173	
Slow	Feb 28, 2017 02:05:47 EDT	http://192.168.11.121:8780/zapstore/trackYourOrder	Page	29548	1780	199	2416	25153	
Slow	Feb 28, 2017 01:25:48 EDT	http://192.168.11.121:8780/zapstore/myorders	Page	29528	1779	199	2414	25136	

Figure 3.227: 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!

To zoom into a particular web site/application in the dashboard, simply click on its name in the dashboard of Figure 3.226. Figure 3.228 will then appear.



Figure 3.228: The RUM dashboard of a chosen web site/application

The dashboard of Figure 3.228 supports an array of icons (see Figure 3.229) in its left panel, which when clicked enables you to review and analyze different facets of real user experience with the chosen web site/application.

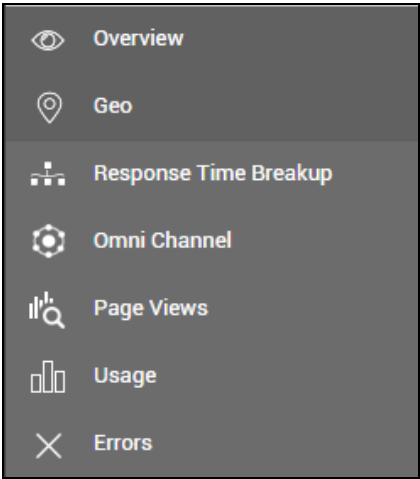


Figure 3.229: The array of icons available in the left panel of the application-specific RUM dashboard

By default, the **Overview** option is selected. As a result, Figure 3.228 appears by default, which provides an overview of the user experience with the target application.

3.6.8.1 Overview of User Experience

The traffic to, the Apdex score of, the responsiveness of, and the number of JavaScript errors (if any) encountered by the target can be quickly ascertained using the **Overview** page (see Figure 3.228). Using the **Response Time Breakup** section in Figure 3.228, you can see, at-a-glance, how many

requests are from different devices (desktop, mobile, etc.) and the response time per device type. This application-level response time analysis provides insights across all pages in the application.

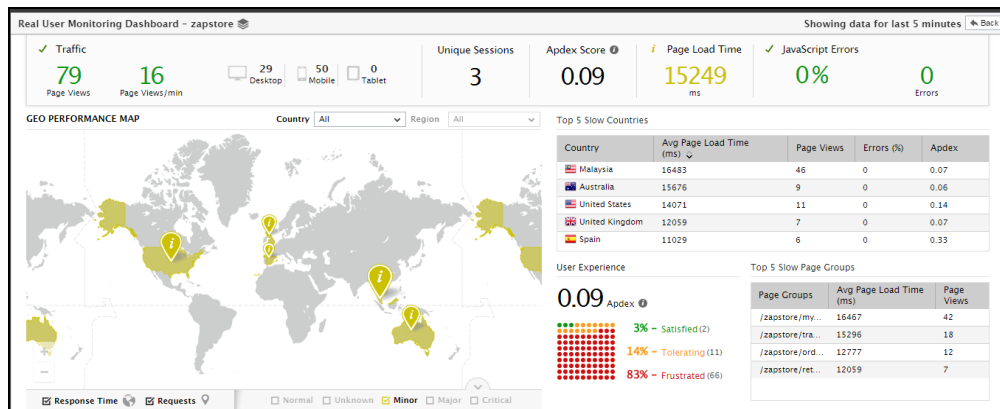


Figure 3.230: The RUM dashboard revealing the user experience per country

The **Overview** also points you to the countries and page groups in which high page load time has been observed.

3.6.8.2 User Experience Analysis Across Geographies

To analyze user experience geographically, click on the **Geo** icon in the array of icons (see Figure 3.229) in the left panel of the dashboard. Figure 3.231 will then appear.

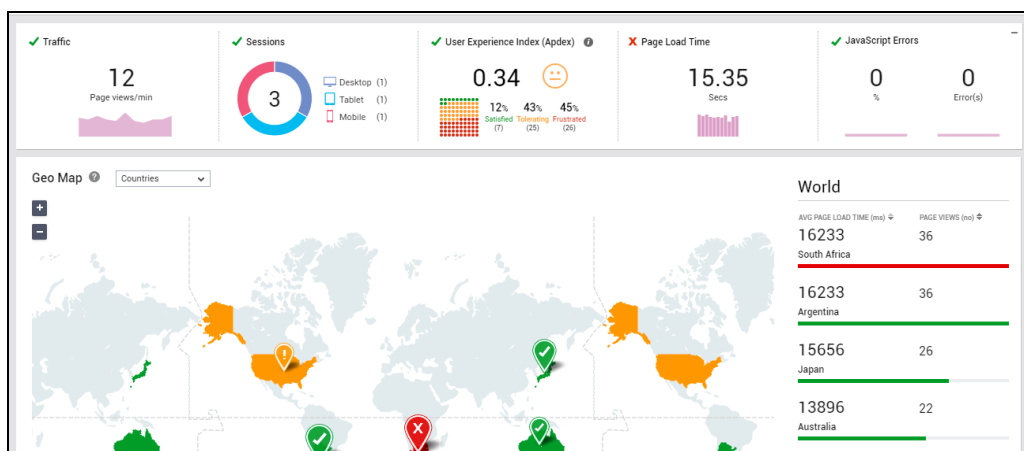


Figure 3.231: User experience analysis across geographies

Without requiring you to pour over tens of metrics, the **Geo Map** in Figure 3.231 quickly tells you where your users are. Location markers accurately pinpoint the countries from which the requests are flowing into the web site / web application. The size of the location marker varies according to the number of requests from each country- i.e., larger the marker, higher will be the number of page

view requests from that country. Also, to indicate the experience of users from a country, both that country and its corresponding marker are colored using conventional color-codes. This way, the map intelligently differentiates between the “happy” countries and the “unhappy” ones.

Note:

At any given point in time, you can choose to view only the request indicators - i.e., the markers - or the response time indicators - i.e., the countries colored using conventional color-codes - in the map. For instance, to hide the markers, click on the  button below the map, and uncheck the **Requests** check box. To hide the colored countries, uncheck the **Response time** check box (see Figure 3.232).

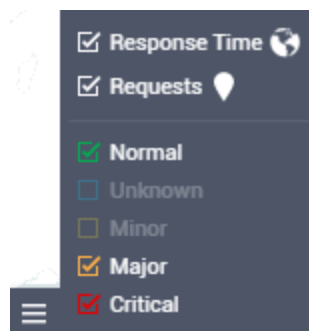


Figure 3.232: Customizing the Geo map

You can also choose to view in the map, only those countries with a poor user experience. For this, you may want to uncheck the **Normal** check box in Figure 3.232, and check the **Major**, **Minor**, **Critical**, and/or **Unknown** check boxes therein.

To know why users from certain countries are 'unhappy' with the web site / web application, click on the problem country in the map. A box depicted by Figure 3.233 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 3.233: Viewing the user experience statistics of a particular country in the geo map

If users from a country are experiencing slowness, it is only natural that you want to figure out whether the problem is widespread or is confined to a specific region within that country. This can be easily determined by zooming into a problem country in the map! For this, just select the country you want to focus on from the **Country** drop-down list in the **Geo Map**. Figure 3.234 will then appear revealing the exact regions (in that country) from which requests are received and the overall experience of the users from each region.

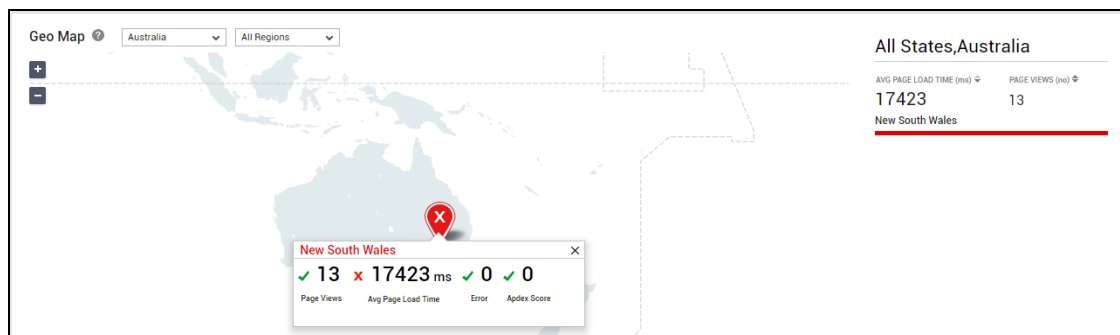


Figure 3.234: Zooming into the regions within a country

Likewise, you can quickly drill-down from a region to view the cities in that region and the quality of their experience with the web site / web application. For this city-level view, select a **Country** and then a **Region** from the **Geo Map**. Figure 3.235 will then appear. Using Figure 3.235, you can precisely pinpoint the city that is experiencing slowness.

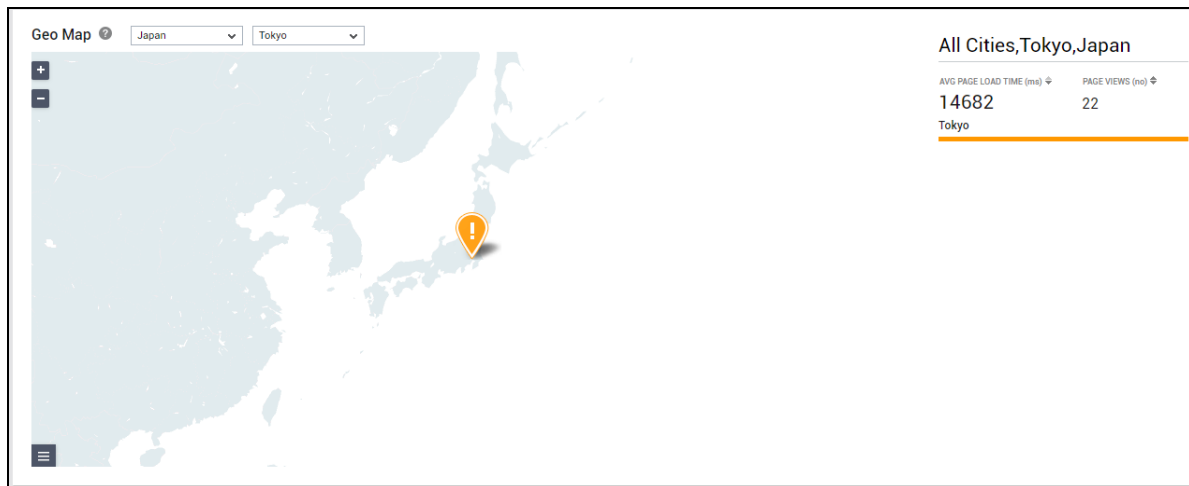


Figure 3.235: Zooming into the cities within a region

If the **Geo Map** reveals that users coming from a particular country/city/region are experiencing slowness, then you may want where such users are seeing the maximum slowness - is it at the browser level? during content download? in the network? or during backend server processing? At the same time, you may want to verify whether the type of device used by the users from a specific country/city/region are impacting their experience with the application, and if so, in what way. The **Response Time Breakup** section in this geographical view enables this dual analysis (see Figure 3.236). The contents of this section will vary depending upon the option you choose from the **Geo Map** drop-down. For instance, if the **Countries** option is chosen from the **Geo Map** drop-down, then the **Response Time Breakup** section will display distribution pie charts revealing where application users from across all the countries in the world are spending their time. Likewise, if you choose a specific **Country** from that drop-down, then the **Response Time Breakup** will provide the breakup of response time for all users from that country alone.

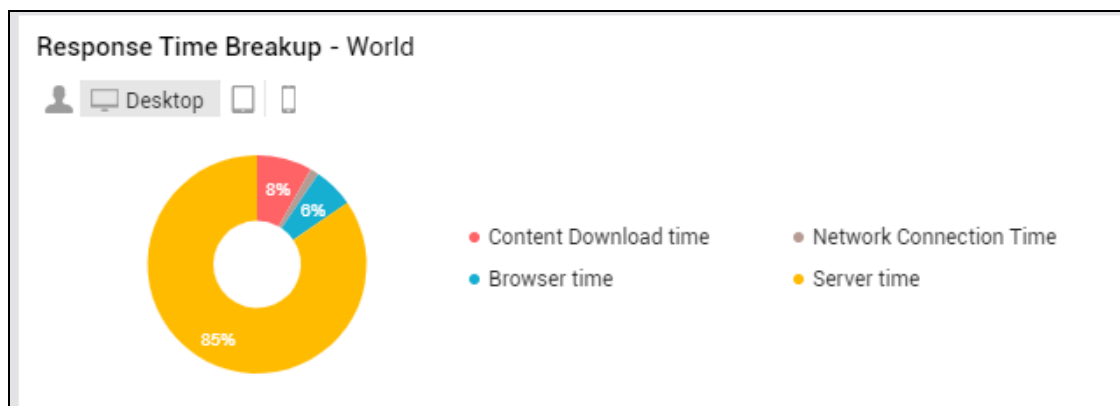


Figure 3.236: Response Time Breakup of users from a chosen geography using a chosen device type

Also, by default, the **Response Time Breakup** section presents the breakup of the response time of all application users, regardless of what type of device they are using. This is why, the **End-user** icon in this section is selected by default. If required, you can use this section to analyze the experience of users depending upon the device type they use. For instance, click on **Desktop** icon in this section to know where desktop users are experiencing a bottleneck (Figure 3.236).

If the **Geo Map** reveals that users coming from a particular country/city/region are experiencing slowness, then you may want to figure out which page groups accessed by the users from that country/city/region are the slowest. This is where the **Top- 5 Slow Page Groups <Countries/Regions/Cities>** list helps. Depending upon the geography you zoom into in your **Geo Map** - whether it is for the world over, a particular country, or a specific region - this list will display the top-5 page groups in that geography in terms of responsiveness. For example, if **Countries** is chosen from the drop-down in your **Geo Map**, then the top-5 list will display the top-5 'least responsive' page groups accessed by application users across the world (see Figure 3.237).

Top 5 Slow Page Groups - World		
PAGE GROUPS	AVG PAGE LOAD TIME (ms)	PAGE VIEWS
/EasyKart/Login.jsp	18289	3
/EasyKart/AddToCart.jsp	17834	8
/EasyKart/BrowseProducts.jsp	17789	10
/EasyKart/PaymentPage.jsp	12860	11
/EasyKart/Welcome.jsp	2428	1

Figure 3.237: Slowest page groups in a chosen geography

3.6.8.3 Response Time Analysis

For a more detailed analysis of response time, click on the **Response Time Breakup** icon in the array of icons (see Figure 3.229) in the left panel of the dashboard. Figure 3.238 will then appear.

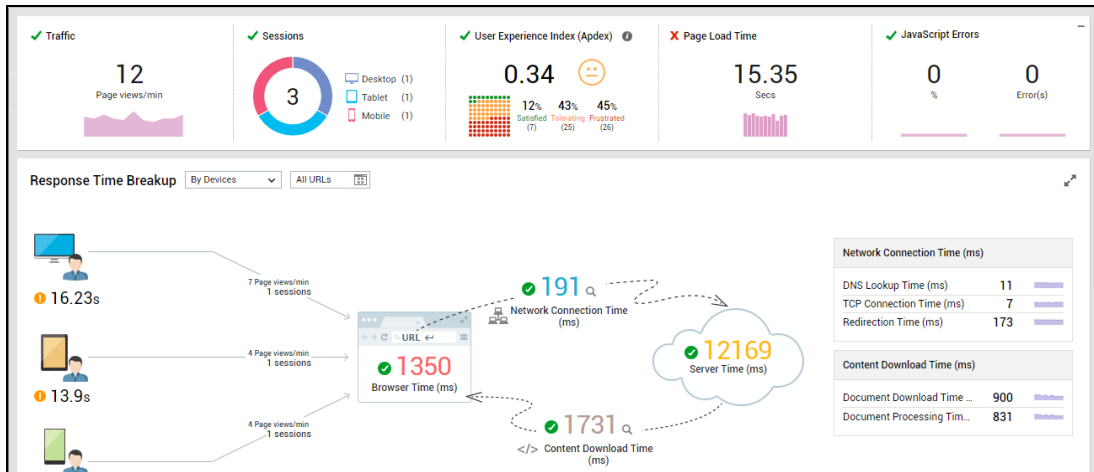


Figure 3.238: Response time analysis

Using the **Response Time Breakup** section in Figure 3.238, you can easily and effectively analyze the response time per device type - i.e., desktops, mobiles, and tablets. A quick look at this visual representation will tell you if users using a specific type of device are seeing more slowness than the others.

You can also compare the responsiveness of a specific URL across device types by selecting the URL of interest to you from the **All URLs** drop-down. This will reveal the devices from which the URL has been accessed and the response time of the URL as seen as by the users of each device (see Figure 3.239). You will thus be able to tell whether the URL was slow across the board, or was slow for the users of certain device types only. This way, you can figure out if the poor application performance that the users of certain device types are experiencing is because the application is not optimized to handle requests for certain URLs from certain device types.

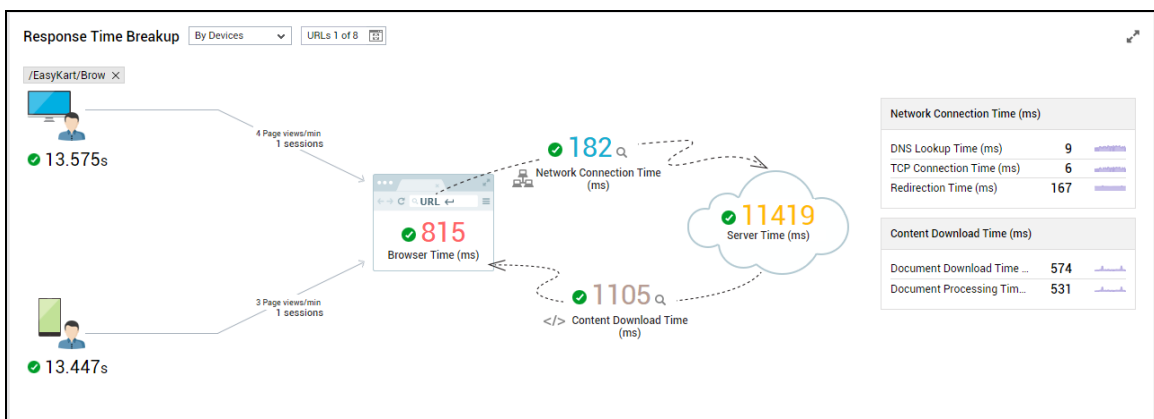


Figure 3.239: Response time breakup for a chosen URL across device types

The **Response Time Breakup** section also reveals the probable cause for poor UX with the application - is it owing to a slowness at the browser end? a latent network? delay in content loading? or a processing bottleneck on the backend server? *Note that this insight is provided at the application-level and not at the device type/URL level.* If the section reveals that high network connection time is the reason for the poor responsiveness of the application, you can click on the 'magnifying glass' icon alongside **Network Connection Time** to view a break-up of that metric. This way, you can accurately diagnose the reason why network connections to the application took too long. Additionally, you can also identify those URLs that were badly hit by high network connection time (see Figure 3.240).

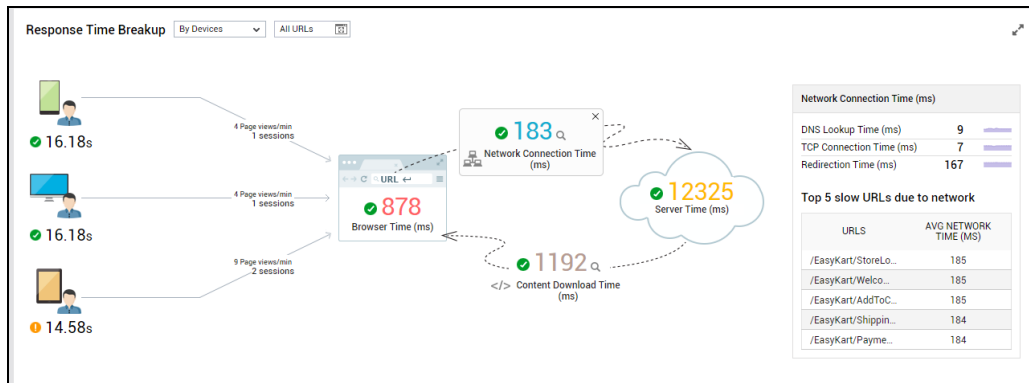


Figure 3.240: Break-up of Network Connection Time and the top-5 URLs impacted by high network connection time

On the other hand, if the section reveals that high content downloading time is the reason for the poor responsiveness of the application, you can click on the 'magnifying glass' icon alongside **Content DownloadTime** to view a break-up of that metric. This way, you can accurately diagnose the reason why content downloading took too long. Additionally, you can also identify those URLs that were badly hit by high content download time (see Figure 3.241).

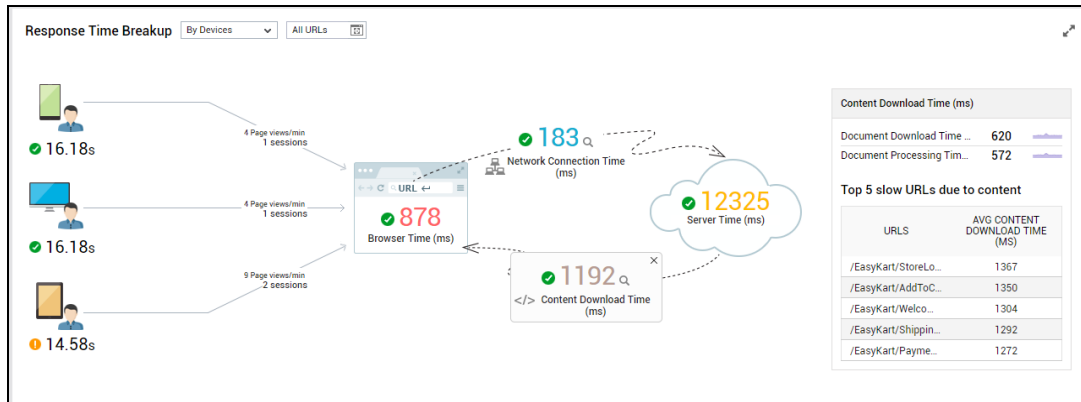


Figure 3.241: Break-up of Content Download Time and the top-5 URLs impacted by high content download time

When viewed together with the per-device response time, this application-level response time breakup also reveals if the slowness at the browser, network, server, and/or content download areas has impacted all users or only users of a few specific device types. Typically, if an application is not optimized to handle requests from certain device types, then such requests may end up spending more time on the browser, network, server, and/or content downloading. Using Figure 3.241 therefore, you can identify those device types that the application is not optimized to support and where such optimizations should be performed.

Similarly, you can analyze responsiveness at the page group-level by selecting the **Page Groups** option from the **Response Time Breakup** drop-down (see Figure 3.242).

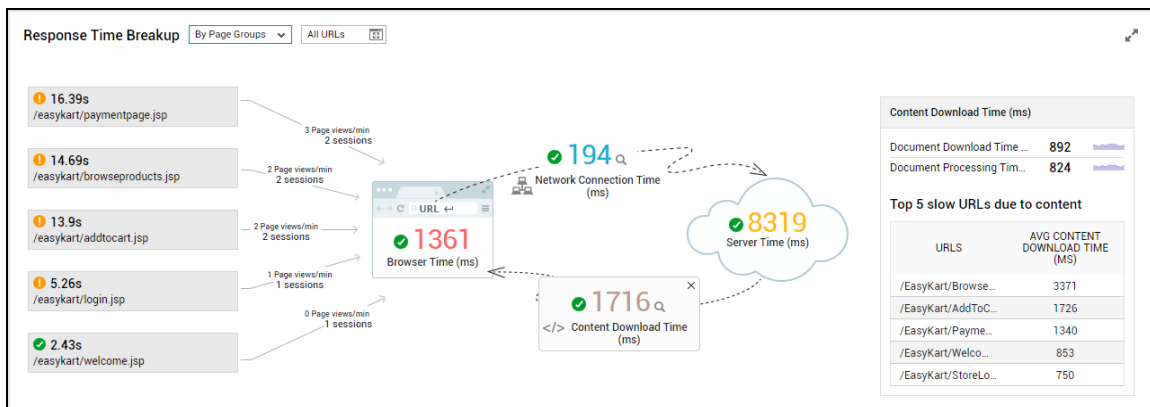


Figure 3.242: Response time breakup analysis by page group

Using Figure 3.242, you can analyze the performance of the top-5 'least responsive' page groups in the target application. The response time of and the load imposed by (page views/minute and current sessions) each page group is displayed therein. This way, you can identify the page groups

with which application users are having the maximum trouble! Here again, the response time break-up (browser time, network time, server time, and content download time) is presented at the application-level (and not at the page group-level). Using the break-up, you can determine why user experience with the application is poor. When correlating the application-level response time breakup with the page group-level analytics, you can figure out if the slowness that users are seeing on the browser, network, content processing, and/or are impacting all application users or only those accessing specific page groups. Sometimes, certain page groups in the application may not be engineered right to handle user requests. This inefficiency may manifest as high browser time, network time, server time, or even content download time. Figure 3.242 will point you to those page groups that may need fine-tuning.

3.6.8.4 User Experience Analysis Across Devices

For detailed analysis of response times across devices, click on the **Omni Channel** icon in the array of icons (see Figure 3.229) in the left panel of the dashboard. Figure 3.243 will then appear.

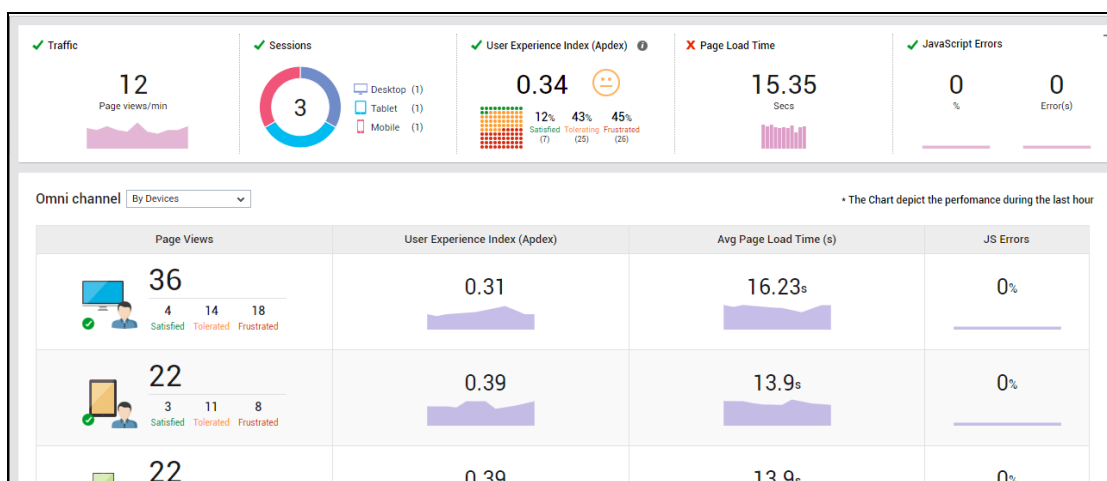


Figure 3.243: User experience analysis by channel

For each type of device that is accessing the target application, the **Omni Channel** section in Figure 3.243 reveals the total count of page view requests that came from that device during the last hour, the Apdex score representing the experience of users using that device, the average page load time as seen by the users of that device during the last hour, and the count of JavaScript errors that were encountered by the users of that device in the last hour. This comprehensive comparison quickly and accurately points you to those types of devices that are experiencing maximum slowness when accessing the application.

3.6.8.5 User Experience Analysis by URL

If you want a quick summary of user experience across URLs or for a specific URL, click on the **Page Views** icon in the array of icons (see Figure 3.229) in the left panel. Figure 3.244 will then appear.

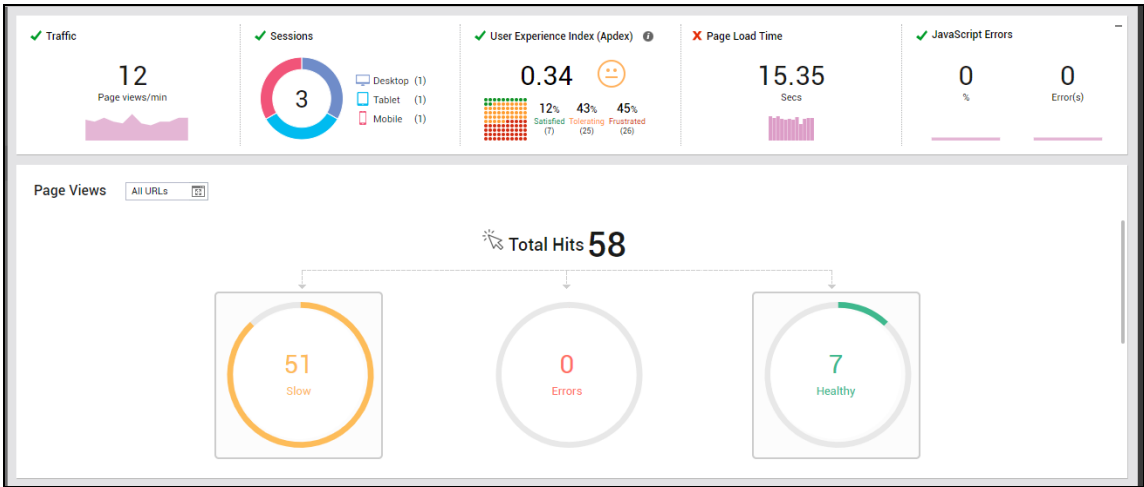


Figure 3.244: Analysis across URLs

By default, the **Page Views** drop-down in Figure 3.244 is set to **All URLs**. Accordingly, this section reveals the total page view requests to the target application and the count of slow, healthy, and error page views. This will reveal if there are any URLs in the target that are loading slowly. If this representation reveals slow page views, then you can scroll down to view a summary of the slow page views, as depicted by Figure 3.245.

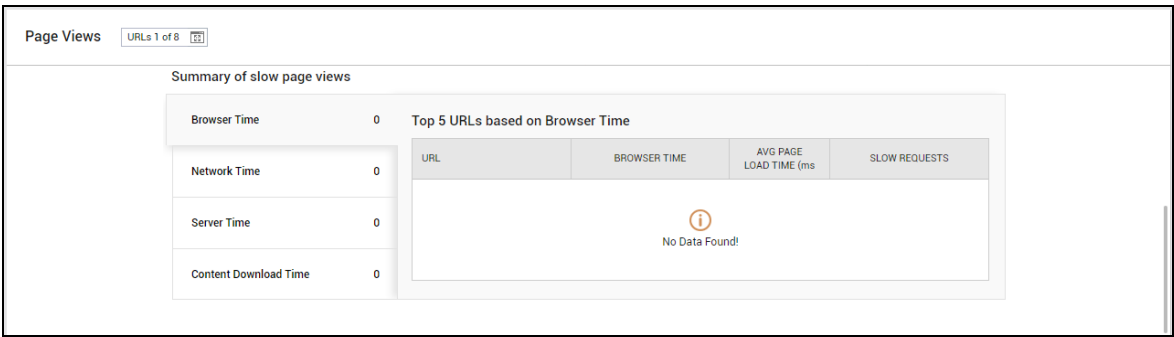


Figure 3.245: Summary of slow page views

In this summary, you will be able to view the break-up of page load time on one side - i.e., browser time, network time, server time, and content download time. Clicking on any component of page load time will list the top-5 URLs that registered the maximum value for that component. In other words, if

say **Browser Time** is chosen, then the top-5 URLs with the highest browser time will be displayed. In the process, you can figure out where your bottlenecks lie, and which URLs are impacted by these bottlenecks.

If you want to focus on the responsiveness of a particular URL, then select that URL from the **Page Views** drop-down in Figure 3.244. The **Page Views** section will then graphically represent the experience of users hitting that URL alone (see Figure 3.246).

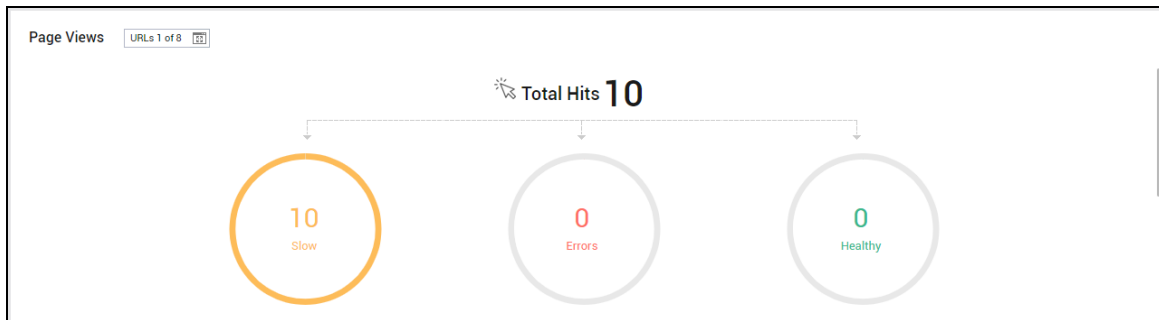


Figure 3.246: Analysis of a chosen URL

3.6.8.6 Analysis by Usage

To cross-correlate across usage analytics related to the target application and to study usage trends, click on the **Usage** icon in the array of icons (see Figure 3.229) in the left panel of the dashboard. Figure 3.247 will then appear.

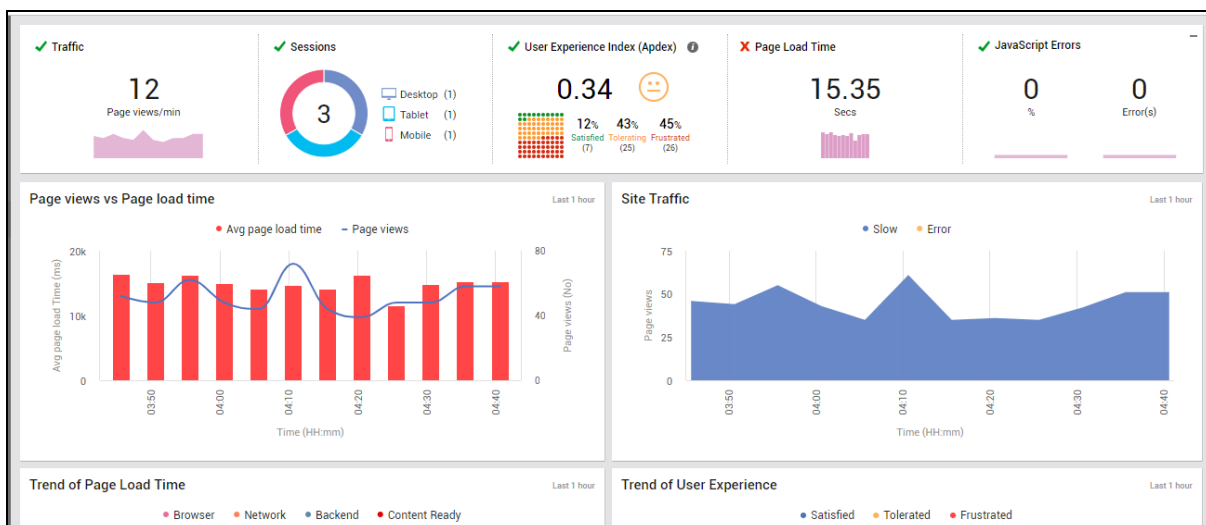


Figure 3.247: Usage analysis

Using the **Page views vs Page load time** graph in Figure 3.247, you can easily time-correlate page view requests with page load time for the last one hour. In the process, you can figure out if page load time increased consistently, or only when page views increased. If the page load time keeps increasing regardless of the count of page views, you may need to investigate the reasons for the same. On the other hand, if application responsiveness deteriorates only when the traffic to the application increases, it could only mean that the application is not optimized to handle heavy loads.

You can use the **Site Traffic** graph in Figure 3.247 to closely analyze the traffic to the target site/application during the last hour. At the end of this analysis, you will be able to tell how healthy the page views were in the last hour - were they slow most of the time? did they experience too many JavaScript errors? or were they healthy most of the time?

You can then scroll down Figure 3.247 to analyze past trends in page load time and user experience, and to view the distribution of page views by device and browser (see Figure 3.248).

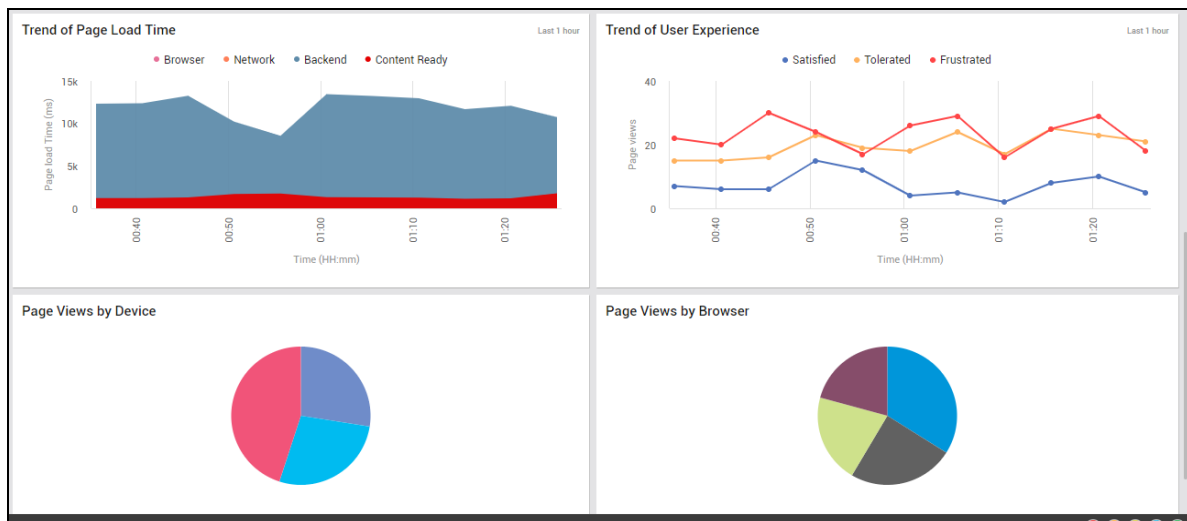


Figure 3.248: Trend and distribution analysis

Using the **Trend of Page Load Time** graph in Figure 3.248, you can understand where your page view requests have spent most of their time in the last one hour - at the browser? on the network? in the backend server? or in content downloading? This will accurately reveal where your bottlenecks lie and where you need to focus your engineering efforts.

You can then analyze user experience trends using the **Trend of User Experience** graph in Figure 3.248. With the help of this graph, you can figure out whether the user experience with your application has been satisfactory or not during the last hour. Disturbing trends in user experience come to light in the process, so you can initiate measures to negate this trend before it becomes worse!

To view the most popular browsers and devices, in terms of the page view requests coming from them, use the **Page Views by Device** and **Page Views by Browser** distribution pie charts in Figure 3.248. To make sure that your application delivers an above-par experience to a majority of your users, optimize your application to offer the best support to these popular browsers and devices.

3.6.8.7 Error Analysis

For insights into JavaScript errors encountered by the target applications, click on the **Errors** icon in the array of icons (see Figure 3.229) in the left panel. Figure 3.249 will then appear.

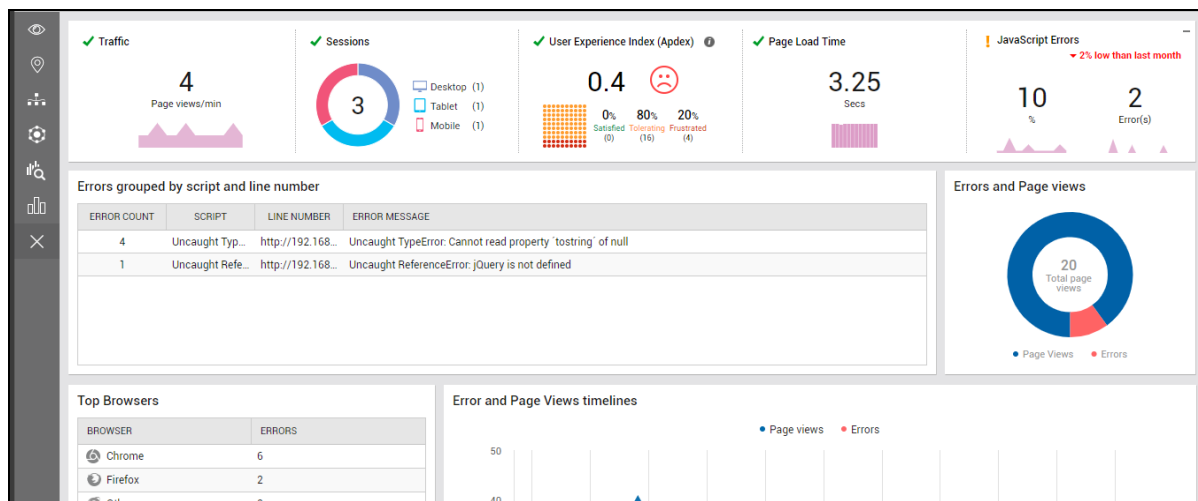


Figure 3.249: Error analysis

To know what JavaScript errors occurred on the target application, take a look at the **Errors grouped by script and line number** section. From this tabulation, you can quickly identify what errors occurred, where (on which line), and in which script.

To determine how many of the pages currently viewed have errors in them, use the **Errors and Page Views** distribution chart. This is also a good indicator of how badly page views are impacted by JavaScript errors.

Scroll down the **Error** analysis view to identify the top browsers and top URLs, in terms of the JavaScript error count (see Figure 3.250).

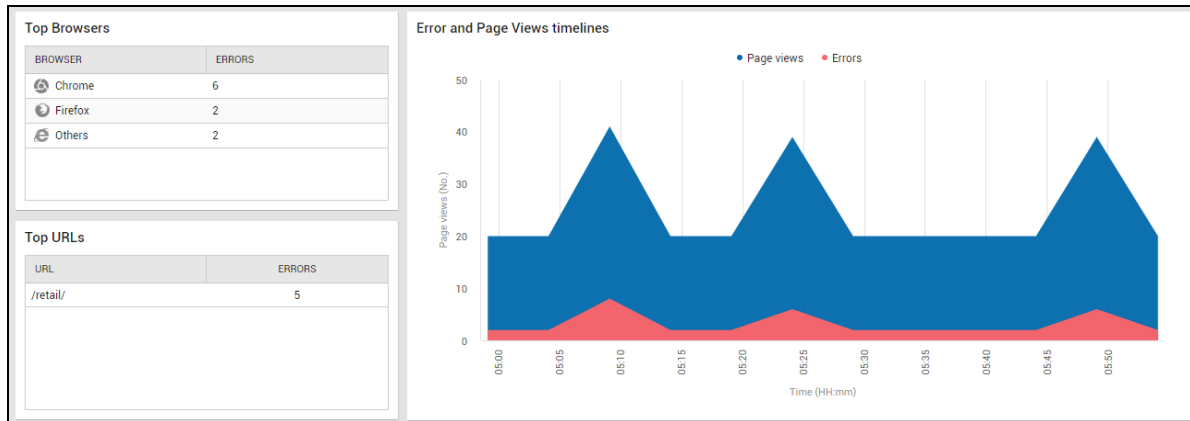


Figure 3.250: Identifying error-prone browsers and URLs, and correlating error views with total page views

Use the **Error and Page Views timelines** chart in Figure 3.250 to study variations in the count of error views and total page views during the last hour. This will give you a fair idea of how frequently JavaScript errors occurred in the target application, and how many page views were impacted by these errors.

Chapter 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

Chapter 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.

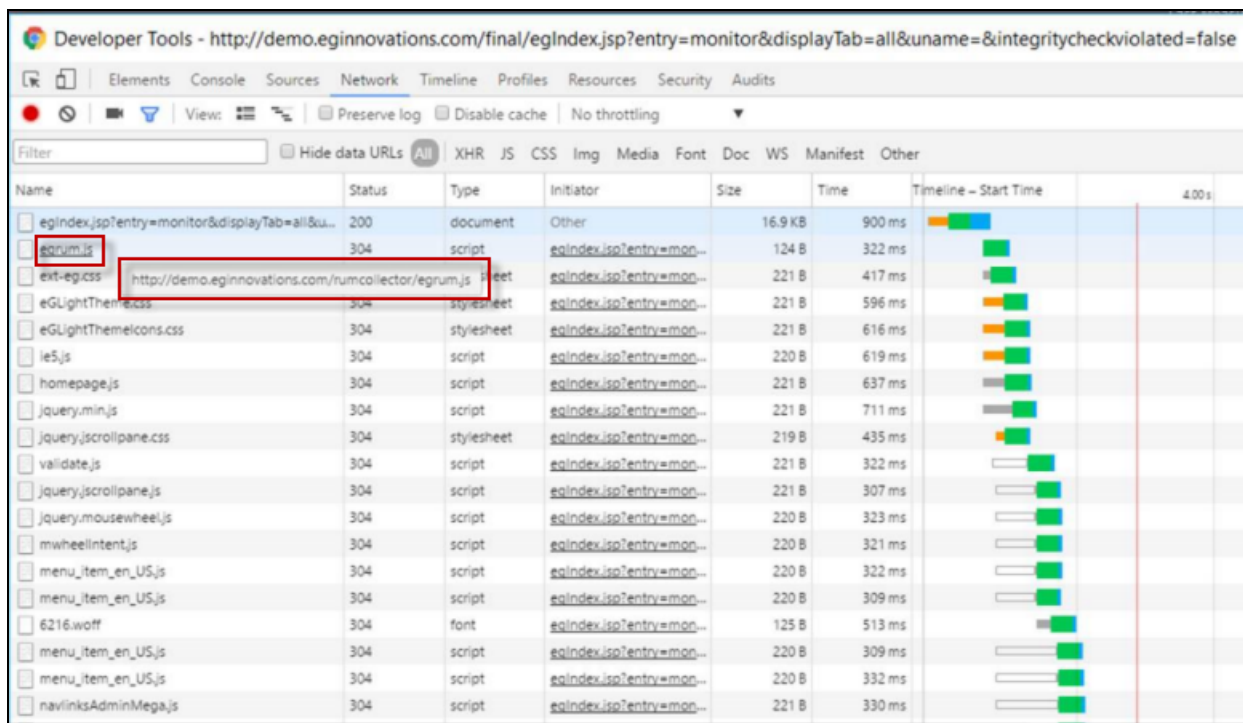


Figure 5.1: Developer Tools console revealing whether/not the JavaScript is successfully inserted in a web page

- 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 it's 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.

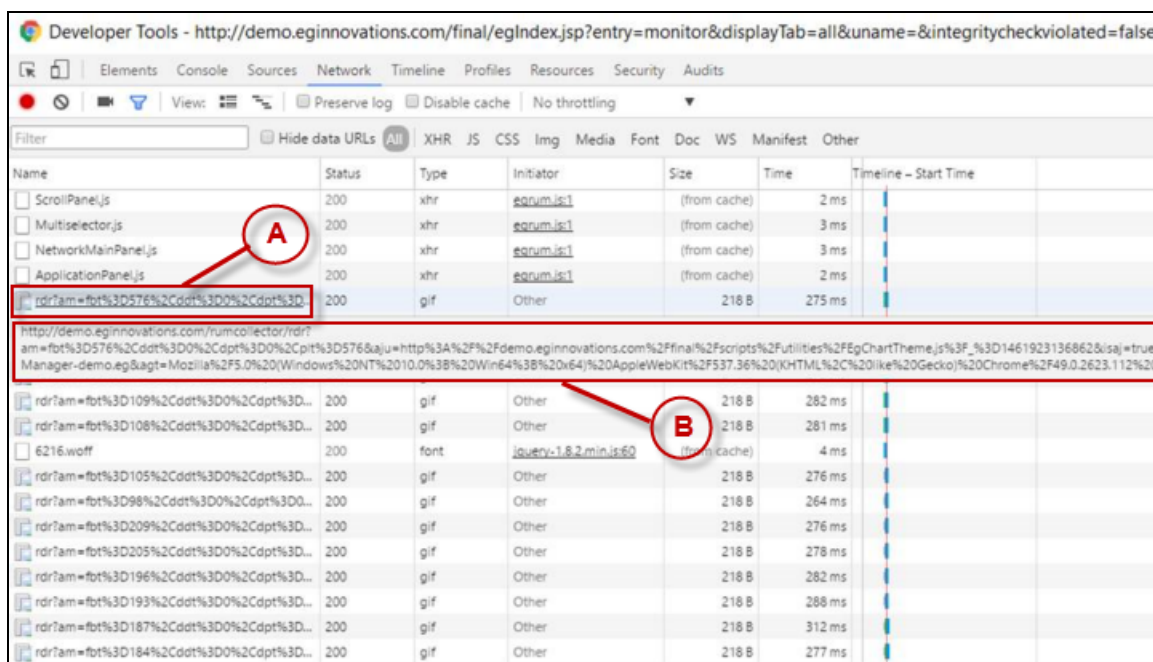


Figure 5.2: Determining whether/not the browser is sending performance beacons to the RUM collector

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_INSTALLDIR>\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.

Chapter 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.

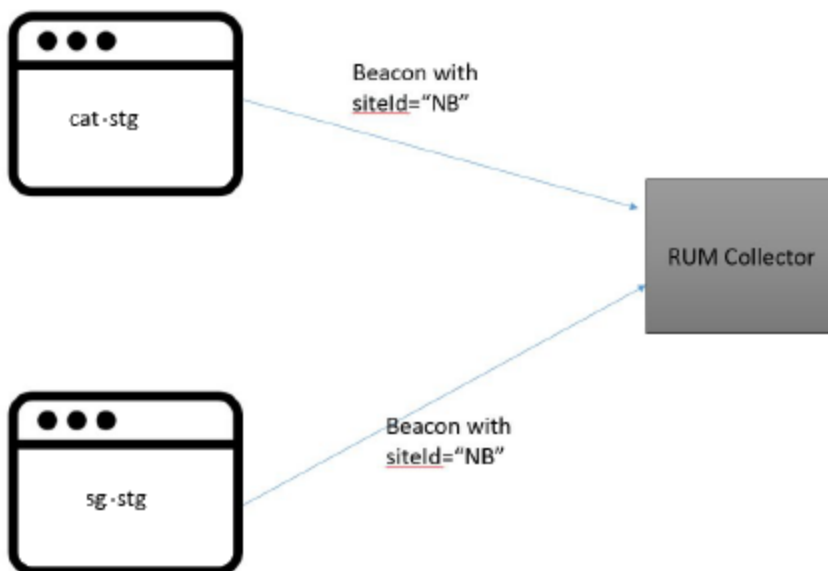


Figure 6.1: Two web sites sending beacons to the RUM collector using a single Site ID

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).

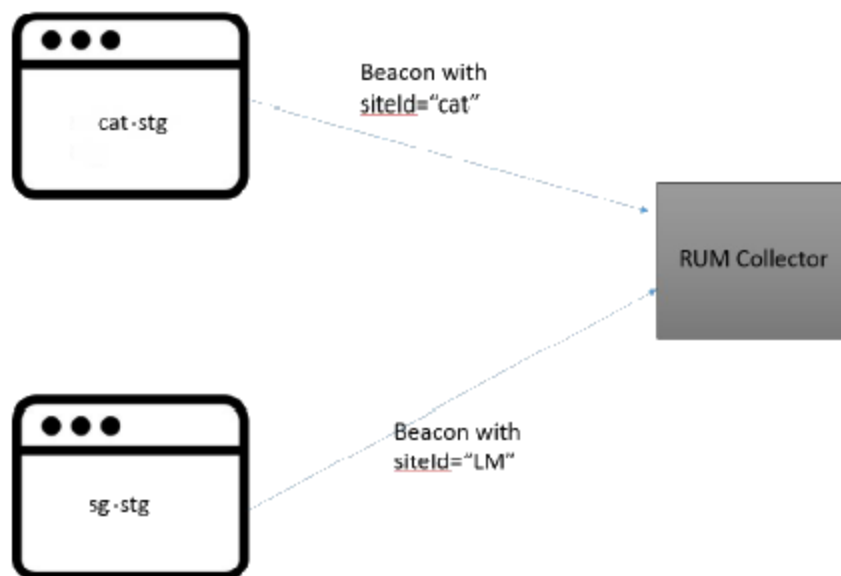


Figure 6.2: Two web sites sending beacons to the RUM collector using different Site IDs

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.

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. **In my environment, I have RUM-enabled a web site on an IIS web server using IIS URL Rewrite rules. This web site services requests from internet users. However, the RUM collector to which the browsers have to send beacons resides in the intranet. What do I do to make sure that the browsers are able to send performance beacons to that RUM collector?**

In this scenario, you will have to enable reverse proxy for IIS using ARR (Application Request Routing), so that performance beacons sent by client browsers to IIS are automatically forwarded by IIS to the RUM collector in an internal network. The steps in this regard are as follows:

- Download the ARR module from the following link:
<https://www.iis.net/downloads/microsoft/application-request-routing>
- Install the ARR module on the IIS web server host.
- You need to separately enable reverse proxy for each web site on IIS - i.e., for each RUM component that you have managed. To achieve this, first open the **Internet Information Services (IIS) Manager**. Expand the Sites node in the tree structure in the left panel of the manager and select the site for which you want to configure reverse proxy. From the options displayed in the right panel, select the **URL Rewrite** icon by double-clicking on it. Once the **URL Rewrite** section opens in the right panel, right-click on the **Inbound Rules** list in that section and pick the **Add Rule** option from the shortcut menu that appears (see Figure 6.3).

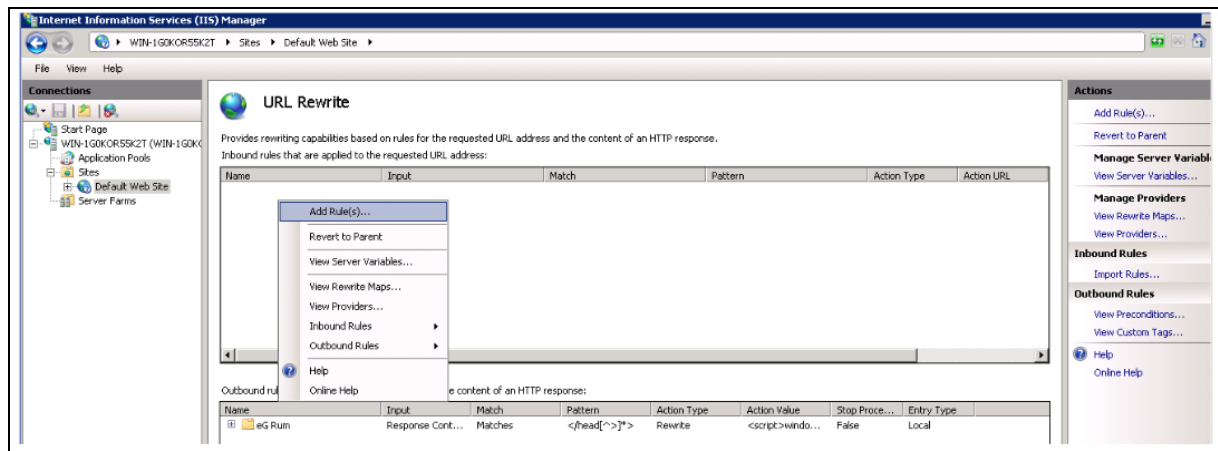


Figure 6.3: Selecting the Add Rule option

- From the **Add Rule(s)** dialog box that then appears (see Figure 6.4), select **Reverse Proxy** as the inbound rule template and click the **OK** button.

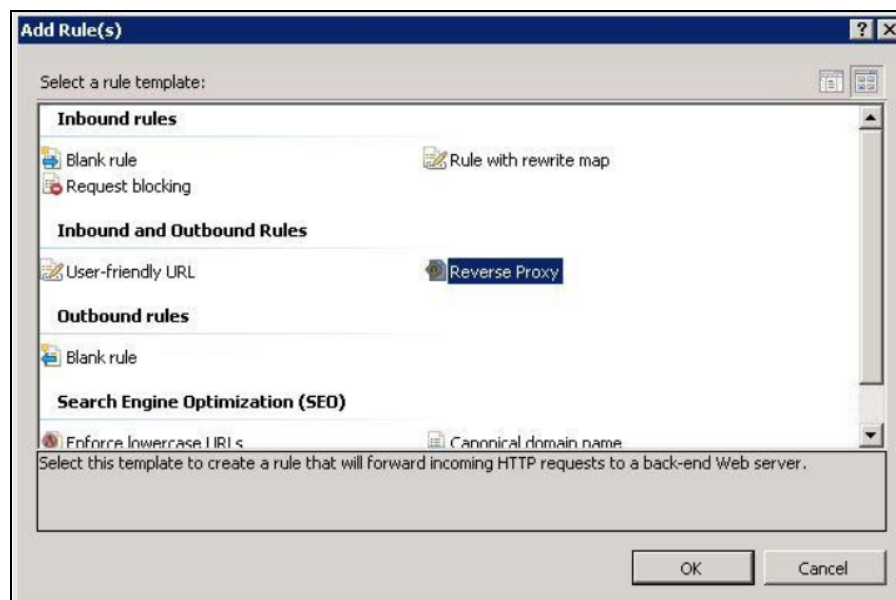
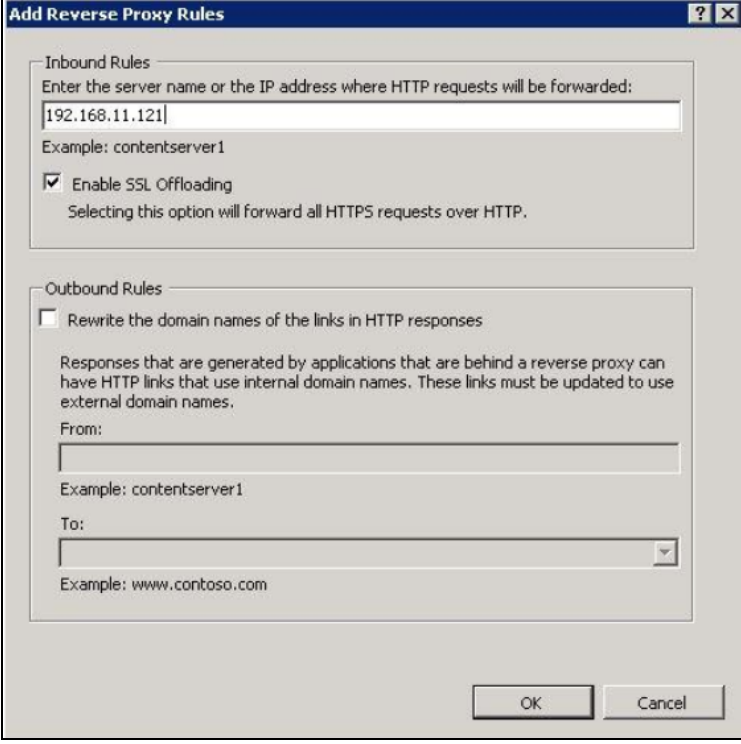


Figure 6.4: Selecting Reverse Proxy as the inbound rule template

- When Figure 6.5 appears, in the Inbound Rules section of Figure 6.5, **Enter the server name or IP address** of the server to which IIS has to redirect requests. In this case, performance beacons from client browsers in the public network should be redirected to the eG RUM collector in an internal network. So, here, specify the name or IP address of the eG RUM collector.



Add Reverse Proxy Rules

Inbound Rules
 Enter the server name or the IP address where HTTP requests will be forwarded:

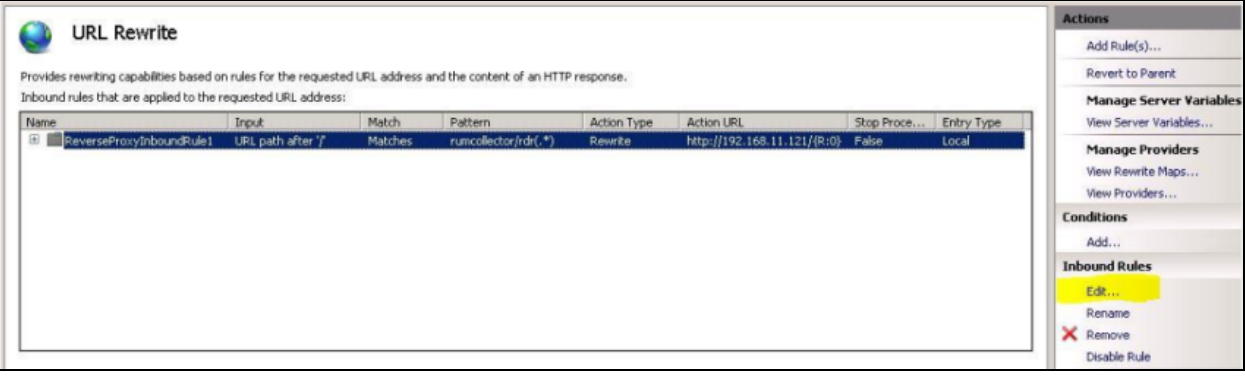
 Example: contentserver1
☒ **Enable SSL Offloading**
 Selecting this option will forward all HTTPS requests over HTTP.

Outbound Rules
☐ **Rewrite the domain names of the links in HTTP responses**
 Responses that are generated by applications that are behind a reverse proxy can have HTTP links that use internal domain names. These links must be updated to use external domain names.
 From:
 Example: contentserver1
 To:
 Example: www.contoso.com

OK Cancel

Figure 6.5: Specifying the server name / IP address of the RUM collector

- Figure 6.6 will then appear, where you will be able to view the inbound rule you just created. Select the new inbound rule and click the **Edit** option in the **Actions** pane alongside (see Figure 6.6).



URL Rewrite
 Provides rewriting capabilities based on rules for the requested URL address and the content of an HTTP response.
 Inbound rules that are applied to the requested URL address:

Name	Input	Match	Pattern	Action Type	Action URL	Stop Proce...	Entry Type
ReverseProxyInboundRule1	URL path after '/'	Matches	runcollector/rdr/(.*)	Rewrite	http://192.168.11.121/(R:0)	False	Local

Actions
 Add Rule(s)...
 Revert to Parent
Manage Server Variables
 View Server Variables...
Manage Providers
 View Rewrite Maps...
 View Providers...
Conditions
 Add...
Inbound Rules
 Edit...
 Rename
 Remove
 Disable Rule

Figure 6.6: Editing the new inbound rule

- In the **Edit Inbound Rule** dialog box that then appears (see Figure 6.7), specify the URI **Pattern** that the IIS web server should route to the eG RUM Collector. Then, in the **Action** section, specify the **Rewrite URL** that the configured URI **Pattern** should be redirected to.

Also, select the **Append query string** check box.

Edit Inbound Rule

Match URL

Requested URL: Using:

Pattern:

☒ Ignore case

Conditions

Server Variables

Action

Action type:

Action Properties

Rewrite URL:

☒ Append query string

☐ Log rewritten URL

☐ Stop processing of subsequent rules

Actions

[Online Help](#)

Figure 6.7: Specifying the URI pattern to capture and the Rewrite URL it should be redirected to

- Finally, click the **Apply** option in the **Actions** pane of Figure 6.7 to apply the changes to the inbound rule.

8. 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.

9. 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.

10. 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.

11. **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.

12. **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.

13. **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, configurable. To configure the limit, do the following:

- Edit the **rum.properties** file in the <EG_RUM_DATA_COLLECTOR_INSTALL_DIR>\tomcat\webapps\rumcollector\WEB-INF\libdirectory (on Windows; on Unix, this will be the /opt/rum/tomcat/webapps/rumcollector/WEB-INF/libdirectory)
- 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.

14. **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.

15. **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.

16. **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.

17. **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.

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

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

19. **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 *eG Manager and Database Sizing Calculator* that is provided to you.

20. **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.

21. **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.

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

No, this is not possible.

23. **Can a single web page in a web site/web application contain multiple RUM scripts? Under what circumstances will this happen?**

Multiple RUM scripts can be injected into a single web page under the following circumstances:

- If you are using two eG managers, and you have managed the same web site/web application as a Real User Monitor component in both the managers using the same SiteID, then a single web page in that web application will be injected with two RUM scripts.
- If you are managing a single web site/web application as two Real User Monitor components using a different SiteID for each, then again, a single web page in that web application will be injected with two RUM scripts.

24. **In the above scenario, where multiple RUM scripts are injected in the same web page, how many times will the browser load the RUM script and how many beacons will be sent to the collector? If there are multiple collectors at play here, which collector will receive the beacons?**

The table below answers your questions.

Scenarios	How many times is the eG RUM script loaded?	How many beacons are sent to the collector?	Which collector receives the beacons?
The same collector is assigned to the same Real User Monitor component in two managers	Once	One	N/A
A single web application is managed as two Real User Monitor components in an eG manager, and both these components are assigned the same collector.	Once	One	N/A
Two different collectors are assigned to the same Real User Monitor component in two managers.	Twice	One	The collector that is assigned first to the Real User Monitor component.
A single web application is managed as two Real	Twice	One	The collector that is assigned first to the

User Monitor components in an eG manager, with each component being assigned a different collector.			Real User Monitor component.

25. **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.

26. **Can I configure high availability for the eG RUM Collector?**

Yes, you can configure high availability for the eG RUM Collector in the same way you configure high availability for the eG agent. The broad steps to achieve this are as follows:

- Prepare two Windows machines with identical specifications
- Both machines must be in the same Active Directory domain, as a best practice. Both the machines should have the same domain role. The recommended role is member server.
- The File Share Server should also be in the same Active Directory domain as both these machines.
- Install an eG RUM Collector on each of the machines.
- Then, proceed to create the Windows cluster with the two machines.
- Next, create a Windows file share from a File share server. This file share will be used as a third vote in the **Node and File Share Majority** quorum mode that is to be set for the cluster.
- Configure cluster quorum settings.
- Add the eG RUM Collector service as a shared cluster resource.

Each of these steps have been elaborately discussed for an eG agent, in the eG Installation Guide. The same detailed procedures apply to the eG RUM Collector as well.

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. **If I have a single RUM collector receiving beacons from multiple sites, can I configure this collector to communicate with multiple remote agents?**

Yes. You can configure a single collector to communicate with multiple remote agents.

3. **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.

4. **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.

5. **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.

6. **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.

7. **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 Chapter 6). In order to do this, we should select the specific site and create rules for it.

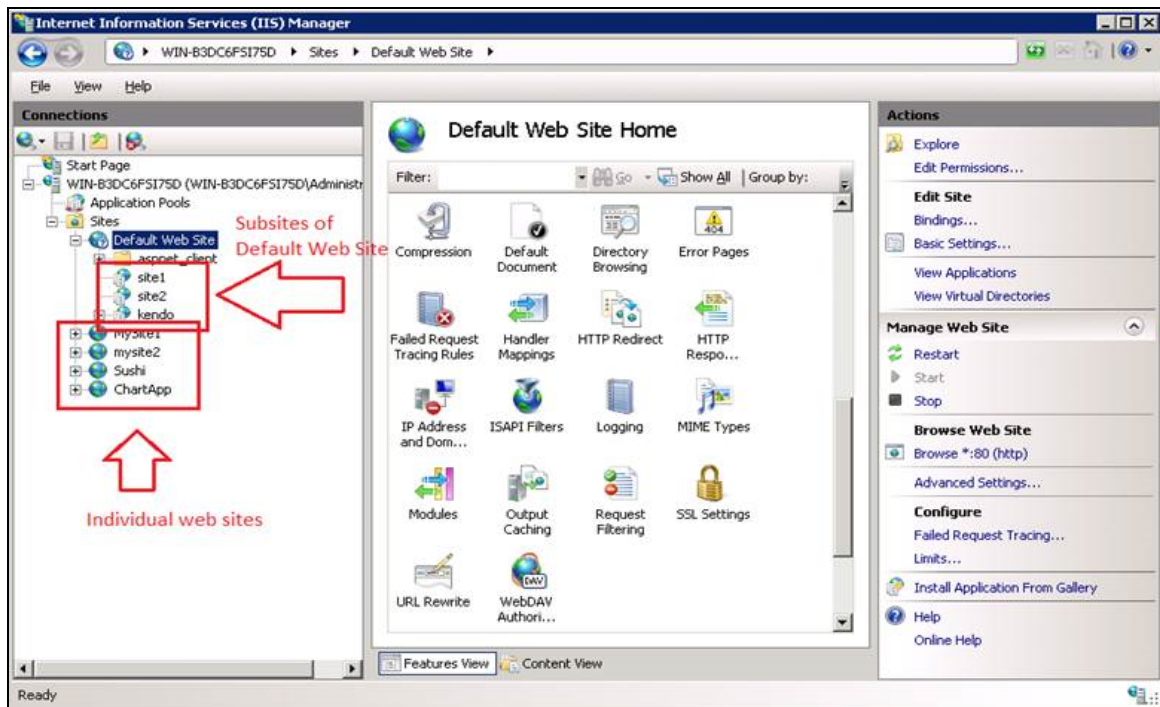
(A) Default Web Site. (the subsites below shares the same hostname and URL pattern alone differs.)

(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)



8. 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 of this document.

9. 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.

10. **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 Instrumenting the Web Site / Web Application to be Monitored topic.

11. **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.

12. **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.

13. **Is the eg_rum.jar already embedded with the Site_name and beacon_URL?**

Yes. Both the Site_name and beacon-url are embedded within the jar/dll file.

14. **If the answer to question 12 is 'yes', that means each egrum.jar file is unique to each website. Is there any other way to each get those egrum.jar from our web page for non-admin user?**

Yes. egrum.jar/Eg.RUM.dll file is unique to each website. The egrum.jar or the Eg.RUM.dll file can only be downloaded from admin interface. We do not have any other page to download these files.

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 user 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.

14. I have seen that in the RUM dashboard, intranet requests do not carry any 'location' reference - i.e., the country, region, and city of these requests are either marked as 'Unknown' or are marked with the default location I have in my configuration. On the other hand, for internet requests, the geographic location of the requests is accurately discovered and displayed in the dashboard. How can I make sure that the RUM dashboard displays the exact geographic location of the intranet requests as well?

To understand how to configure geographic locations in the RUM dashboard for intranet requests, let us consider an example. Assume that your enterprise spans three different buildings, namely - North Building, South Building, and East Building. Each building is in a different network. Users from these buildings communicate with your web site over the intranet.

1. The first step is to obtain the IP range of each of the buildings from your network/infra teams.
2. Next, edit the **geolocation-local-map.xml** file in the <EG_MANAGER_INSTALL_DIR>\manager\tomcat\webapps\rumcollector\WEB-INF\lib directory. Here, configure the IP range of each of your networks. This can be performed in one of the following ways:

- By configuring the starting IP and ending IP address of each range, OR;
 - By configuring the starting IP address and the sub-net mask of each range.
3. If you know the starting and ending IP addresses of every network, then go with the first option - i.e., configure the starting and ending IP addresses of every network in the **geolocation-local-map.xml** file to indicate the geographic location. In this case, follow the steps below.
- Assume that the starting and ending IP addresses of every building in our example is as follows:

Starting IP address	Ending IP address	Building
192.168.8.1	192.168.8.250	North Building
192.168.9.1	192.168.9.250	South Building
192.168.10.1	192.168.10.250	East Building

- Next, for each building, configure a 'location' block in the XML file. This means that three location blocks will have to be configured in the XML file. In the **<country-code>** tag of every location block, make sure you specify a unique code to represent the building to which that location block corresponds. For the purpose of our example, let us use the code NB, SB, and EB to represent the North Building, South Building, and East Building:

```
<location start-ip-address="192.168.8.1" end-ip-address="192.168.8.250">
```

```
<country>Singapore</country>
```

```
<region>Cross Street</region>
```

```
<city>Singapore</city>
```

```
<country-code>NB</country-code>
```

```
<latitude>1.2896700</latitude>
```

```
<longitude>103.8500700</longitude>
```

```
</location>
```

```
<location start-ip-address="192.168.9.1" end-ip-address="192.168.9.250">
```

```
<country>Singapore</country>
```

```
<region>Cross Street</region>
```

```
<city>Singapore</city>
```

```

<country-code>SB</country-code>

<latitude>1.2896700</latitude>

<longitude>103.8500700</longitude>

</location>

<location start-ip-address="192.168.10.1" end-ip-address="192.168.10.250">

<country>Singapore</country>

<region>Cross Street</region>

<city>Singapore</city>

<country-code>EB</country-code>

<latitude>1.2896700</latitude>

<longitude>103.8500700</longitude>

</location>

```

As you can see, the <country>, <region>, <city> tags carry the same information in all the 3 <location> blocks in our example. This is because, all 3 buildings in our example are in the same country, region, and city. Since the <latitude> and <longitude> denote the co-ordinates of the country, even they remain the same in all 3 <location> blocks.

- Finally, save the XML file.
4. On the other hand, if you do not know the ending IP addresses, then you can have the eG Enterprise system automatically compute the IP address range of each of your networks. For this, you need to go with the second option - i.e., configure the starting IP address and sub-net mask of each of your networks. In this case, follow the steps below:

- Assume that the starting IP address and subnet-mask of each building in our example is as follows:

Network IP Range	Subnet-mask	Building
192.168.8.0	255.255.254.0	North Building
192.168.9.0	255.255.254.0	South Building
192.168.10.0	255.255.254.0	East Building

- Now, edit the **geolocation-local-map.xml** file in the <EG_MANAGER_INSTALL_DIR>\manager\tomcat\webapps\rumcollector\WEB-INF\lib directory. For each building

in our example, configure a 'location' block in the XML file in the following format.

```
<location network="192.168.8.0" subnet-mask="255.255.254.0">
<country>Singapore</country>
<region>Cross Street</region>
<city>Singapore</city>
<country-code>NB</country-code>
<latitude>1.2896700</latitude>
<longitude>103.8500700</longitude>
</location>

<location network="192.168.9.0" subnet-mask="255.255.9.0">
<country>Singapore</country>
<region>Cross Street</region>
<city>Singapore</city>
<country-code>SB</country-code>
<latitude>1.2896700</latitude>
<longitude>103.8500700</longitude>
</location>

<location network="192.168.10.0" subnet-mask="255.255.10.0">
<country>Singapore</country>
<region>Cross Street</region>
<city>Singapore</city>
<country-code>EB</country-code>
<latitude>1.2896700</latitude>
<longitude>103.8500700</longitude>
</location>
```

As you can see, the <country>, <region>, <city> tags carry the same information in all the 3 <location> blocks in our example. This is because, all 3 buildings in our example are in the same country, region, and city. Since the <latitude> and <longitude> denote the co-ordinates of the country, even they remain the same in all 3 <location> blocks. However, note that the <country-code> tag has been configured with a unique code representing

each building in our example, namely - NB for North Building, SB for South Building, and EB for East Building.

- Finally, save the XML file.

5. Then, edit the **eg_geodetails.ini** file in the <EG_INSTALL_DIR>\manager\config directory.
6. In the **[COUNTRY_NAME_TO_CODE_MAPPING]** section of the file, insert an entry to map the country code specified within the <country-code> tag of the <location> block in the **geolocation-local-map.xml** file, with a country name - in the case of our example - the building name. The format of this specification is: <CountryName>=<Country-code>. For the purpose of our example, 3 entries of the aforesaid format will have to be inserted in the **[COUNTRY_NAME_TO_CODE_MAPPING]** section. Sample entries are provided below:

```
North Building=NB
```

```
South Building=SB
```

```
East Building=EB
```

7. Next, map the new country-code with the corresponding latitude and longitude markings. This is required to indicate where in the geo-location map of the RUM dashboard, the marker should rest to pinpoint this location. For this, insert an entry of the format: <Country-code, as specified in the XML file>=<Latitude>|<Longitude>, in the **[COUNTRY_CENTER_LAT_LNG]** section of the **eg_geodetails.ini** file. In the case of our example, 3 entries will have to be made in this section. Sample entries are provided below:

```
NB=1.352083|103.819836
```

```
SB=1.252308|100.813768
```

```
EB=1.208245|101.88564
```

8. Finally, save the file.
 9. Then, copy the **eg_geodetails.ini** file on the eG manager host to the <EG_AGENT_INSTALL_DIR>\agent\config directory on the eG agent host.
 10. Restart the eG manager and agent.
15. **Where users connect to a target web site via multiple telecom lines, can eG RUM pinpoint the telecom line that was used by the user who often complained of bad experience?**

eGRUM captures and reports the Client IP address of each RUM page view, as part of detailed diagnostics. You can use this IP address to identify the telecom line manually.

About eG Innovations

eG Innovations provides intelligent performance management solutions that automate and dramatically accelerate the discovery, diagnosis, and resolution of IT performance issues in on-premises, cloud and hybrid environments. Where traditional monitoring tools often fail to provide insight into the performance drivers of business services and user experience, eG Innovations provides total performance visibility across every layer and every tier of the IT infrastructure that supports the business service chain. From desktops to applications, from servers to network and storage, from virtualization to cloud, eG Innovations helps companies proactively discover, instantly diagnose, and rapidly resolve even the most challenging performance and user experience issues.

eG Innovations is dedicated to helping businesses across the globe transform IT service delivery into a competitive advantage and a center for productivity, growth and profit. Many of the world's largest businesses use eG Enterprise to enhance IT service performance, increase operational efficiency, ensure IT effectiveness and deliver on the ROI promise of transformational IT investments across physical, virtual and cloud environments.

To learn more visit www.eginnovations.com.

Contact Us

For support queries, email support@eginnovations.com.

To contact eG Innovations sales team, email sales@eginnovations.com.

Copyright © 2020 eG Innovations Inc. All rights reserved.

This document may not be reproduced by any means nor modified, decompiled, disassembled, published or distributed, in whole or in part, or translated to any electronic medium or other means without the prior written consent of eG Innovations. eG Innovations makes no warranty of any kind with regard to the software and documentation, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The information contained in this document is subject to change without notice.

All right, title, and interest in and to the software and documentation are and shall remain the exclusive property of eG Innovations. All trademarks, marked and not marked, are the property of their respective owners. Specifications subject to change without notice.