eG Java Business Transaction Monitoring

eG Innovations Product Documentation

www.eginnovations.com

Total Performance Visibility

Table of Contents

Table of Figures

Chapter 1: Introduction

A business transaction represents a type of user request to a web application. For instance, the following types of requests are considered business transactions for an online retail banking application:

- Logging in
- Balance checking
- Funds transfer
- Bill payments
- Logging out

User experience with a web application not only relies on the successful completion of these user requests/transactions, but also on their rapid execution. This is why, even if a single transaction slows down, stalls, or fails, user dissatisfaction with the web application as a whole grows. This in turn may cause user complaints to increase, support costs to sky rocket, and revenues to dip.

To avoid such disastrous results, web application administrators should monitor every business transaction closely and promptly identify the slow/stalled/failed transactions. Most importantly, administrators will have to determine where and why these transactions under-performed – i.e., identify the root-cause of poor transaction performance - so that the problem can be quickly resolved before users begin doubting the stability of the web application.

Root-cause isolation is often the most challenging! This is because, most web applications these days overlay multi-tier environments characterized by multiple application servers, database servers, remote services, etc. Every business transaction to such web applications travels through multiple nodes, using remote calls to external services, to fulfill its purpose. For example, an online transaction to shop for goods may access a ShopCart web page on a web server. Every time an item is added to a shopping cart, the web server may make an HTTP/S call to a web application server to invoke the business logic. The business logic may then make a database call to run a query for retrieving the total count of goods that that user has shopped for so far. A slowdown in even one node or a delay in processing even a single remote service call can impact the performance of the transaction. To accurately isolate where the actual bottleneck lies, administrators should employ an APM solution that can trace the entire path of every business transaction, measure the total round-trip time of each transaction, identify the synchronous and asynchronous calls made by the transaction at various nodes, and compute the time spent by the transaction at each node, for each call. This can be achieved using the **eG Java Business Transaction Monitor (BTM)**.

1.1 The eG Java Business Transaction Monitor (BTM)

The **eG Java BTM** employs an advanced 'tag-and-follow' technique to trace the complete path of each business transaction to a web application, end-to-end. When doing so, it auto-discovers the application servers the transaction travels through, and also automatically ascertains what remote service calls were made by the transaction when communicating with the servers. In the process, the eG Java BTM measures the following:

- The total response time of each transaction;
- The time spent by the transaction on each application server;
- The time spent by the transaction for processing every external service call (including SQL queries);

Using these analytics, the eG Java BTM precisely pinpoints the slow, stalled, and failed transactions to the web application, enables administrators to accurately isolate where – i.e., on which application server – the transaction was bottlenecked, and helps them figure out exactly what caused the bottleneck – an inefficient or errored query to the database? A slow HTTP/S call to another application server? a time-consuming POJO / JMX method execution? a slow SAP JCO/async call? By quickly leading administrators to the source of transaction failures and delays, the eG Java BTM facilitates rapid problem resolution, which in turn results in the low downtime of and high user satisfaction with the web application.

1.2 Pre- requisites for Java Business Transaction Monitoring Using eG Enterprise

The following are the pre-requisites for performing Java business transaction monitoring using eG:

- For the eG Java Business Transaction Monitor to function, your eG Enterprise infrastructure should include:
  - An eG Manager of version 6.2.0 (or above)
  - eG Agents of version 6.2.0 (or above)
  - An eG database on a Microsoft SQL Server 2008 (or above) (OR) An Oracle Database Server 9i (or above)
- The **eG Java Business Transaction Monitor (BTM)** can be installed on Java containers only - i.e., Java applications / J2EE-enabled web, application, and messaging servers. The details are as follows:

Supported JVMs

The eG Java BTM can be installed on JVM 1.6 (and above) only, regardless of the JVM vendor. JVM 1.4 or below is not supported. Only partial support is available to JVM 1.5; this is because, in JVM 1.5, cross-application transaction tracing cannot be performed since the HttpURLConnection class cannot be byte-code instrumented to perform cross-JVM tag-and-follow.

Vendor-specific JVM support is as detailed below:

- Oracle Hotspot JVM 1.6 to 12
- BEA JRockit 1.6
- IBM JVM 1.6 to 1.8
- OpenJDK 1.6 to 12
- SAP JVM 1.6 to 1.8
- Azul Zing 1.6 to 1.8
- Azul Zulu 1.6 to 1.8

Supported Application Types

Java-based application types

Supported/Unsupported Application Servers

Application Server	Supported Versions	Unsupported Versions
WebSphere Application Server	7.x, 8.x, 9.x	6.x and below
WebSphere Liberty Profile	8.x	
WebLogic Server	9.x, 10.x, 12.x	8.x and below
JBoss	4.x, 5.x	
JBoss AS/EAP	6, 7, 7.x	
WildFly	8.x, 9.x, 10.x to 18	
Tomcat	5.x, 6.x, 7.x, 8.x, and 9.x	4.x and below
TomEE	7.x, 8.x	
Jetty	9.x	

GlassFish	3.x, 4.x, 5.x	2.x
Payara	4.x, 5.x	
Oracle Application Server (OC4J)		
Spring Boot Application with Embedded Tomcat	2.0 to 2.2	
Spring Boot Application with Embedded Jetty	2.0 to 2.2	
Spring Boot Application with Embedded Undertow	2.0 to 2.2	
Any Custom Web Application with Embedded Tomcat	6.x to 9.x	

Supported Packaged Applications

- Oracle Peoplesoft
- Oracle JDEdwards
- Liferay (with Tomcat) 6.2 to 7.3

Supported Entry Pointcuts

- Servlets/Filters 2.5 (and above)
- JSPs
- Struts 1.x, 2.x
- Spring MVC 3.x and 4.x
- Web Services - SOAP and REST
- Java Server Faces (JSF) 1.x and 2.x
- JMS 1.x and 2.x

Supported HTTP Exit Pointcuts

- HTTPURLConnection
- ApacheHTTPClient 3.x and 4.x

Supported Web Service Exit Points

- SOAP based web services
- Axis 1.x and 2.x
- Apache CXF
- Resteasy
- Jersey

Supported Databases

- Oracle 8i, 9i, 10g, 11g, 12c
- IBM DB2 9.x
- MS SQL Server 2005, 2008, 2012, 2016
- PostgreSQL 8.x and 9.x
- MySQL 5.x and 8.0
- HSQLDB
- MariaDB 5.x
- IBM Informix
- Sybase
- SAP HANA
- AWS RDS
- AWS Aurora
- Mongo DB 3.x

Supported Database/ORM Frameworks

- iBATIS/MyBatis 2.x and 3.x
- Hibernate
- JPA

Supported Database Clusters

- Microsoft SQL Cluster
- Oracle RAC

Note:

- In the cross-applications transaction topology flow map, a Microsoft SQL cluster (if any) will be represented only as a standalone Microsoft SQL database server.
- Oracle RAC supports a variety of JDBC URL formats. The eG Java BTM currently captures only those queries issued to an Oracle RAC for just a subset of these JDBC URL formats. If a Java transaction issues a query to an Oracle RAC for one of the supported JDBC URL formats, that Oracle RAC component will only be represented as a standalone Oracle database server in the cross-application transaction topology flow map.

Supported ESB and Integration Frameworks

- Mule ESB 3.9
- Apache Camel (Only JMS integration)

Supported Drivers

- Oracle- Thin
- DB2
- Microsoft SQL Server
- Connector/J
- jTDS - Type4
- JDBC2, JDBC2 EE, JDBC3, JDBC4

Messaging Exit Pointcuts

- ActiveMQ 5.x
- JBoss Messaging and HornetQ from JBoss
- IBM MQ
- JMS Queues and Topics

Middleware Exit Pointcuts

- RMI using JRMP
- EJB - Stateless session bean (SLSB)
- EJB - Stateful session bean (SFSB)

- Runtime Exec (Process Exec)
- LDAP
- Java Mail
- SAP JCO
- JOLT

Caching Frameworks or In-memory Databases

- H2
- HSQL DB
- EHCACHE 2.x
- Redis

Elastic Environments

- Standalone Docker
- Standalone Kubernetes
- AWS ECS (using EC2 Instances)
- AWS EKS (using EC2 Instances)

- The **eG Java Business Transaction Monitor (BTM)** can be installed on only those Java containers that use JDK 1.5 or higher.
- Do not install the **eG Java Business Transaction Monitor (BTM)** on a Java container that is already JTM-enabled.
- For cross application transaction tracing to occur, the Java application being monitored should run only on JRE 1.6 (or higher).
- For complete visibility into the transaction path, make sure that you:
  - BTM-enable each JVM node in the transaction path;
  - Manage each JVM node as a separate component in eG;

1.3 How does the eG Java BTM Work?

To be able to track the live transactions to a web application, eG Enterprise requires that a special **eG Java Transaction Profiler** be deployed on every JVM node (i.e., web ccserver instance) through which the transaction travels. The steps for deployment are discussed in Installing and Configuring eG Java BTM.

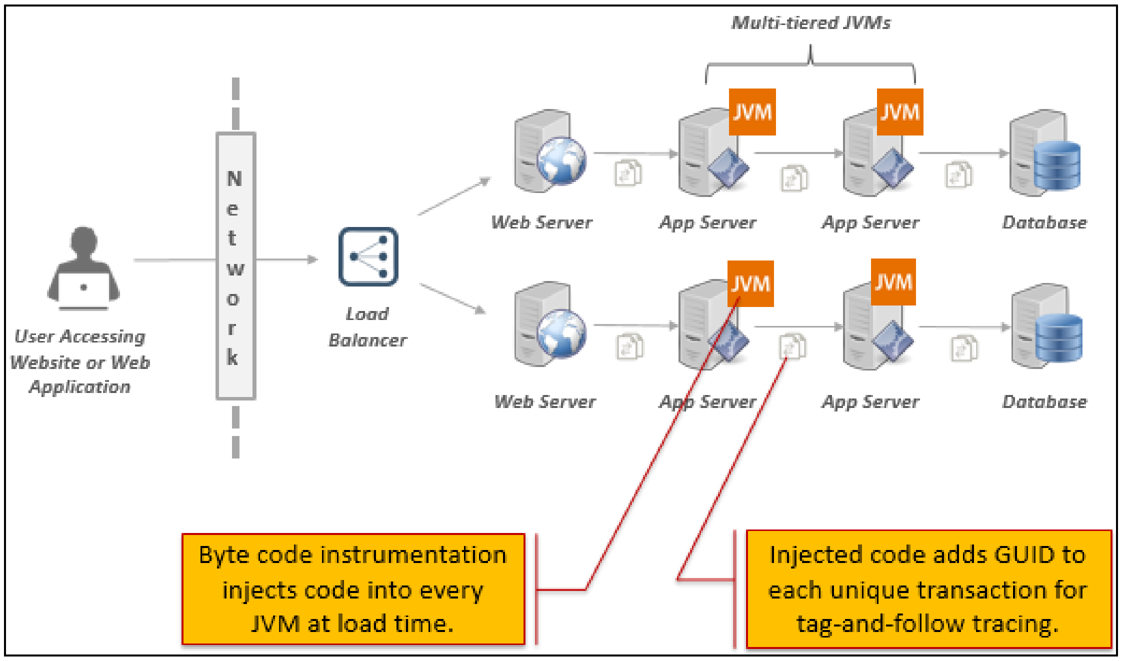

Multi-tiered JVMs
JVM
JVM
JVM
JVM
Web Server
App Server
App Server
Database
Web Server
App Server
App Server
Database
Network
Load Balancer
User Accessing Website or Web Application
Byte code instrumentation injects code into every JVM at load time.
Injected code adds GUID to each unique transaction for tag-and-follow tracing.

Figure 1.1: How eG BTM Works?

The eG Java Transaction Profiler uses byte-code instrumentation to trace transaction path and measure responsiveness. Using this instrumentation mechanism, the profiler injects Java code into the JVM on which it is deployed, at load time. The injected code adds a GUID to each unique transaction on a JVM, so that its path can be accurately traced . In addition, the profiler performs the following tasks for every unique transaction on a JVM:

- Tracks requests to that transaction;
- Measures the average responsiveness of that transaction to the requests;
- Identifies the slow, stalled, and error transactions, and computes the count of such transactions;
- Ascertains the exit calls made by the transaction, the destination of the calls, and measures the time taken by each call;
- Stores all the aforesaid statistics in memory

The profiler then sends all these statistics to the eG agent. To know how and when the profiler transmits metrics to the eG agent, refer to Section **1.4**.

The eG agent deployed on a remote host or on the BTM-enabled JVM periodically runs a **Java Business Transactions** test. This test communicates with the profiler via a configured BTM port, pulls the metrics stored in memory, and reports them to the eG manager for display in the eG monitoring console.

1.4 How does the eG Java BTM Communicate with the eG Agent?

The eG agent should be deployed on the JVM that hosts the eG Java BTM. The eG Java BTM communicates with the eG agent via port number 13700 by default. You can change the default port by following the steps below:

- Edit the eg_tests.ini file (in the <EG_MANAGER_INSTALL_DIR>\manager\config directory on a Windows manager, or the /opt/egurkha/manager/config directory on a Unix manager).
- Configure the new port number against the **AgentServerSocketPortNo** parameter in the [AGENT_SOCKET_SETTINGS] section of the file. Note that you cannot set any random port number against the **AgentServerSocketPortNo** parameter. You have to pick a port number from the list of port numbers present against the **AgentServerSocketPortOrder** parameter, and configure the **AgentServerSocketPortNo** with that port number only. If you configure a port number that is not available against **AgentServerSocketPortOrder**, then the eG Java BTM will not be able to communicate with the eG agent.
- Finally, save the file.

Typically, once the eG agent is configured with the details of the web site to be monitored, the eG Java BTM contacts the eG agent and downloads these details from it.

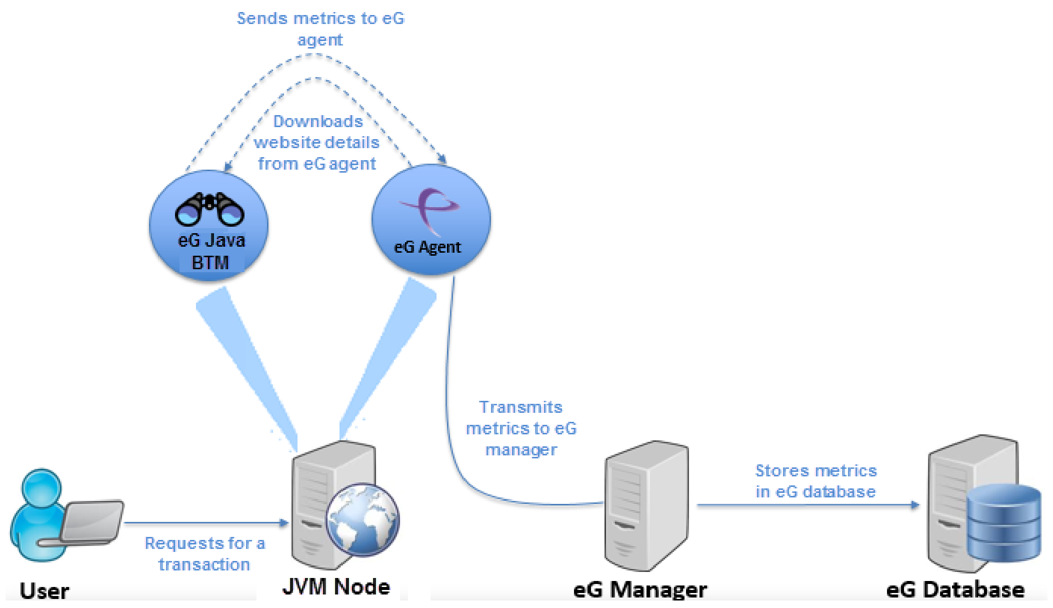

Sends metrics to eG agent
Downloads website details from eG agent
eG Java BTM
eG Agent
Transmits metrics to eG manager
Stores metrics in eG database
Requests for a transaction
User
JVM Node
eG Manager
eG Database

Figure 1.2: Communication between the .NET Profiler and the eG Agent

Then, when a transaction request for the web application comes in, the eG Java BTM injects a code in the application code to trace the path of that request. In the process, the eG Java BTM also collects response time metrics related to that transaction. Every 10 seconds, the eG Java BTM sends these metrics to the eG agent. The eG agent stores these metrics in memory, until the next time it runs the Java Business Transactions test. When the test is run, the agent pulls the metrics stored in memory and sends it to the eG manager.

1.5 Performance Overhead of the eG Java Business Transaction Monitor

eG BTM leaves a very minimal resource footprint on the application it monitors. Typically, it adds a mere 2-5% to the application overhead.

Chapter 2: Installing and Configuring eG Java BTM

The first step towards business transaction monitoring is to BTM-enable the JVM nodes in the transaction path. For this purpose, eG Enterprise requires that a special **eG Application Server Agent** be deployed on every JVM node (i.e., web application server instance) through which the transaction travels.

The **eG Application Server Agent** is available as a file named **eg_btm.jar** on the eG agent host, which has to be copied to the system hosting the application servers being monitored. You then need to configure the application server with the path to the **eg_btm.jar** file to fully BTM-enable the server.

The detailed steps for deployment on different web application servers have been discussed in this section.

2.1 Installing eG Java BTM on a Generic JVM Node

The steps for deploying an eG Java BTM on a JVM node will differ based on the platform on which the target JVM node is running - whether on a Windows platform or a Unix platform.

2.1.1 BTM-Enabling a Generic JVM Node Running on a Windows Platform

If the JVM node is running on a Windows operating system, then follow the steps below to BTM-enable that node:

1. Login to the JVM node.
2. Open a browser on the node, connect to the eG manager, and login to the eG admin interface.
3. Manage the JVM node as a separate component using the eG administrative interface. When managing, make a note of the **Nick name** and **Port number** that you provide.
4. If multiple JVM instances are operating on a single node, and you want to BTM-enable all the instances, then you will have to manage each instance as a separate component using the eG administrative interface. When doing so, make a note of the **Nick name** and **Port number** using which you managed each instance.
5. Next, log out of the eG admin interface. Then, create a **btm** directory anywhere on the JVM node - say, **C:\btm**. Under this directory, create a sub-folder. Make sure that you name this sub-folder in the following format: *<Managed_Component_NickName>_<Managed_Component_Port>*.

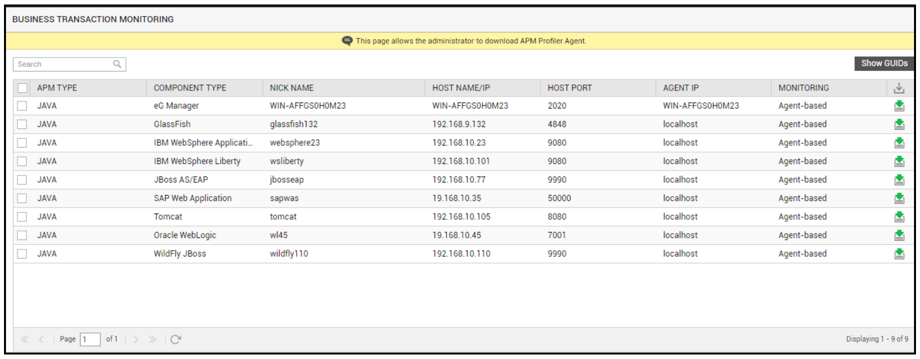
For instance, if you have managed the JVM node using the nick name *AppServer1* and the port number *8088*, the new directory under the **btm** directory should be named as *AppServer1_8088*.

6. If you have managed multiple JVM instances running on a single node, then you will have to create multiple sub-directories under the **btm** directory- one each for every instance. Each of these sub-directories should be named after the **Nick name** and **Port number** using which the corresponding instance has been managed in eG.

7. Once the new sub-directory(ies) is created, open a browser on the JVM node, connect to the eG manager, and login to the eG admin interface again.

8. Follow the Agents -> BTM Profiler Download menu sequence in the eG admin interface.

9. Figure 2.1 will appear listing the servers that can be instrumented for APM by eG. In this list, locate the JVM node that you want to BTM-enable. Once you locate the node, click the **Download** icon corresponding to that node to download the **APM Profiler Agent** to that node. If multiple JVM instances on a single node are managed, then you will have to download the APM Profiler Agent separately for each of the managed instances.


BUSINESS TRANSACTION MONITORING

This page allows the administrator to download APM Profiler Agent.

APM TYPE	COMPONENT TYPE	NICK NAME	HOST NAME/IP	HOST PORT	AGENT IP	MONITORING
JAVA	eG Manager	WIN-AFFGS0H0M23	WIN-AFFGS0H0M23	2020	WIN-AFFGS0H0M23	Agent-based
JAVA	GlassFish	glassfish132	192.168.9.132	4848	localhost	Agent-based
JAVA	IBM WebSphere Applicati...	websphere23	192.168.10.23	9080	localhost	Agent-based
JAVA	IBM WebSphere Liberty	wsliberty	192.168.10.101	9080	localhost	Agent-based
JAVA	JBoss AS/EAP	jbosseap	192.168.10.77	9990	localhost	Agent-based
JAVA	SAP Web Application	sapwas	19.168.10.35	50000	localhost	Agent-based
JAVA	Tomcat	tomcat	192.168.10.105	8080	localhost	Agent-based
JAVA	Oracle WebLogic	wl45	19.168.10.45	7001	localhost	Agent-based
JAVA	WildFly JBoss	wildfly110	192.168.10.110	9990	localhost	Agent-based

Displaying 1 - 9 of 9

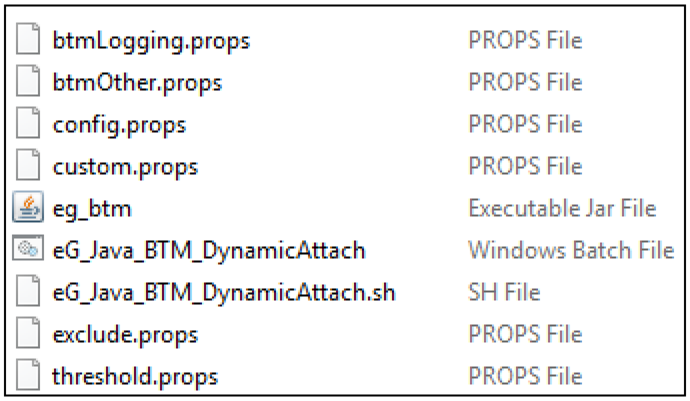
Figure 2.1: Downloading the APM Profiler Agent for the JVM node

10. Upon clicking the **Download** icon in Figure 2.1, a zip file named **javaagent_<Nick_name_of_ JVM_node>_<Port_number_of_JVM_node** will get downloaded. For instance, if you have managed the JVM node using the nickname 'AppServer1' and the port number '8088', then the name of the zip file will be *javaagent_AppServer1_8088*. Where multiple JVM instances have been managed, you will be downloading multiple zip files - one each for every JVM instance. The names of these zip files will automatically carry the nick name and port number you assigned to the corresponding JVM instance.

11. Copy the downloaded zip file(s) to the corresponding sub-directory(ies) of the **btm** directory (see step 6 above). For example, the zip file named *javaagent_AppServer1_8088*, should be copied to the **C:\btm\AppServer1_8088** directory .

12. Extract the contents of each zip file into the same sub-directory to which that zip file was copied.

13. Figure 2.2 depicts the extracted contents of the zip file.



btmLogging.props	PROPS File
btmOther.props	PROPS File
config.props	PROPS File
custom.props	PROPS File
eg_btm	Executable Jar File
eG_Java_BTM_DynamicAttach	Windows Batch File
eG_Java_BTM_DynamicAttach.sh	SH File
exclude.props	PROPS File
threshold.props	PROPS File

Figure 2.2: Contents of the APM Profiler Agent zip

14. From Figure 2.2, it is evident that the zip file contains an **eg_btm.jar** file and a few property files, namely - **btmOther.props** , **btmLogging.props** , **config.props** , **custom.props** , **exclude.props**, and **threshold.props** files.

15. Next, proceed to BTM-enable the JVM node/instance. For that, edit the **btmOther.props** file in the sub-directory (of the **btm** directory) that corresponds to that JVM node/ instance. You will find the following lines in the file:

```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Below property is BTM Server Socket Port, through which eG Agent Communicates
# Restart is required, if any changes in this property
# Default port is "13931"
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#
BTM_Port=13931
#
```

By default, the **BTMPort** parameter is set to 13931. If you want to enable eG Java BTM on a different port, then specify the same here. In this case, when configuring the **Java Business Transactions** test or the **Key Java Business Transactions** test for that application server, make sure you configure the **BTM** port parameter of the test with this port number.

Note:

When BTM-enabling multiple instances on the same server, make sure you configure a different **BTM Port** for each instance.

Also, by default, the **Designated_Agent** parameter will be empty; do not disturb this default setting. In this case therefore, the eG Java BTM will treat the host from which the very first 'measure request' comes in as the **Designated_Agent**.

Note:

In case a specific **Designated_Agent** is not provided, and the eG Java BTM treats the host from which the very first 'measure request' comes in as the **Designated_Agent**, then if such a **Designated_Agent** is stopped or uninstalled for any reason, the eG Java BTM will wait for a maximum of 10 measure periods for that 'deemed' **Designated_Agent** to request for metrics. If no requests come in for 10 consecutive measure periods, then the eG Java BTM will begin responding to 'measure requests' coming in from any other eG agent.

16. Finally, save the **btmOther.props** file.
17. Then, proceed to edit the start-up script of the JVM node/instance being monitored, and append the following lines to it:

```
-DEG_PROPS_HOME=<<PATH OF THE SUB-DIRECTORY CONTAINING THE .PROPS FILES>>
```

```
"-javaagent:<<PATH OF THE eg_btm.jar FILE>>"
```

For instance, if the eg_btm.jar and .props files have been copied to the C:\btm\AppServer1_8088 directory, the above specification will be:

```
-DEG_PROPS_HOME=C:\btm\AppServer1_8088
```

```
"-javaagent:C:\btm\AppServer1_8088\eg_btm.jar"
```

Note:

The "-javaagent…" entry above should be added as one of the JVM options in the start-up script.

18. Finally, save the file, and restart the JVM node.
19. If multiple JVM instances on a single node are monitored, make sure you follow steps 15-18 for each JVM instance.

2.1.2 BTM-Enabling a Generic JVM Node Running on a Unix Platform

If the target JVM node is running on a Unix operating system, then follow the steps below to BTM-enable that node:

1. Login to any system in your environment that supports a browser and has network access to the eG manager.
2. Open a browser on the system, connect to the eG manager, and login to the eG admin interface.
3. Manage the target JVM node as a separate component using the eG administrative interface. When managing, make a note of the **Nick name** and **Port number** that you provide.
4. If multiple JVM instances are operating on a single node, and you want to monitor each of those instances, then you will have to manage each instance as a separate component using the eG administrative interface. When doing so, make a note of the **Nick name** and **Port number** using which you managed each instance.
5. Next, log out of the eG admin interface and the system.
6. Log into the target JVM node. Then, create a **btm** directory anywhere on the target JVM node - say, **/opt/btm**. Under this directory, create a sub-folder. Make sure that you name this sub-folder in the following format: <*Managed_Component_NickName*>_<*Managed_Component_Port*>. For instance, if you have managed the JVM node using the nick name *AppServer1* and the port number *8088*, the new directory under the **btm** directory should be named as *AppServer1_8088*.
7. If you have managed multiple JVM instances running on a single node, then you will have to create multiple sub-directories under the **btm** directory- one each for every instance. Each of these sub-directories should be named after the **Nick name** and **Port number** using which the corresponding instance has been managed in eG.
8. Once the new sub-directory(ies) is created, log out of the JVM node. Log back into the system you used in step 1 above. Open a browser on the system, connect to the eG manager, and login to the eG admin interface again.
9. Follow the Agents -> BTM Profiler Download menu sequence in the eG admin interface.
10. Figure 2.1 will appear listing the servers that can be instrumented for APM by eG. In this list, locate the JVM node that you want to BTM-enable. Once you locate the node, click the **Download** icon corresponding to that node to download the **APM Profiler Agent** to that node. If multiple JVM instances on a single node are managed, then you will have to download the APM Profiler Agent separately for each of the managed instances.
11. Upon clicking the **Download** icon in Figure 2.1, a zip file named **javaagent_<Nick_name_of_ JVM_node>_<Port_number_of_JVM_node** will get downloaded. For instance, if you have managed the JVM node using the nickname 'AppServer1' and the port number '8088', then the name of the zip file will be *javaagent_AppServer1_8088*. Where multiple JVM instances have been managed, you will be downloading multiple zip files - one each for every JVM instance. The

names of these zip files will automatically carry the nick name and port number you assigned to the corresponding JVM instance.

12. Next, using FTP tools like putty or WinSCP, transfer the downloaded zip file (s) to the corresponding sub-directory(ies) (of the **btm** directory), which you created on the JVM node at steps 6 and 7 above. For example, the zip file named *javaagent_AppServer1_8088*, should be transferred to the**/opt/btm/AppServer1_8088** directory on the target JVM node.
13. Log out of the system and log back into the JVM node.
14. Extract the contents of each zip file into the same sub-directory to which that zip file was copied.
15. Figure 2.2 depicts the extracted contents of the zip file.
16. From Figure 2.2, it is evident that the zip file contains an **eg_btm.jar** file and a few property files, namely - **btmOther.props** , **btmLogging.props** , **config.props** , **custom.props** , **exclude.props**, and **threshold.props** files.
17. Next, proceed to BTM-enable the JVM node/instance. For that, edit the **btmOther.props** file in the sub-directory (of the **btm** directory) that corresponds to that JVM node/ instance. You will find the following lines in the file:

```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Below property is BTM Server Socket Port, through which eG Agent Communicates
# Restart is required, if any changes in this property
# Default port is "13931"
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#
BTM_Port=13931
#
```

By default, the **BTMPort** parameter is set to 13931. If you want to enable eG Java BTM on a different port, then specify the same here. In this case, when configuring the **Java Business Transactions** test or the **Key Java Business Transactions** test for that application server, make sure you configure the **BTM** port parameter of the test with this port number.

Note:

When BTM-enabling multiple instances on the same server, make sure you configure a different **BTM Port** for each instance.

Also, by default, the **Designated_Agent** parameter will be empty; do not disturb this default setting. In this case therefore, the eG Java BTM will treat the host from which the very first 'measure request' comes in as the **Designated_Agent**.

Note:

In case a specific **Designated_Agent** is not provided, and the eG Java BTM treats the host from which the very first 'measure request' comes in as the **Designated_Agent**, then if such a **Designated_Agent** is stopped or uninstalled for any reason, the eG Java BTM will wait for a maximum of 10 measure periods for that 'deemed' **Designated_Agent** to request for metrics. If no requests come in for 10 consecutive measure periods, then the eG Java BTM will begin responding to 'measure requests' coming in from any other eG agent.

18. Finally, save the **btmOther.props** file.

19. Then, proceed to edit the start-up script of the JVM node/instance being monitored, and append the following lines to it:

```
-DEG_PROPS_HOME=<<PATH OF THE SUB-DIRECTORY CONTAINING THE .PROPS FILES>>
```

```
"-javaagent:<<PATH OF THE eg_btm.jar FILE>>”
```

For instance, if the eg_btm.jar and .props files have been copied to the /opt/btm/AppServer1_ 8088 directory, the above specification will be:

```
-DEG_PROPS_HOME=/opt/btm/AppServer1_8088
```

```
"-javaagent:/opt/btm/AppServer1_8088\eg_btm.jar”
```

Note:

- The “-javaagent…” entry above should be added as one of the JVM options in the start-up script.
- Also, in Unix environments, when using the agent-based approach, both the agent and the JVM instance will be running on the same host using different user privileges. In this situation, by default, the eG Java BTM logs will not be created. In order to create the same, insert the following entry after the -DEG_PROPS_HOME specification.

```
-DEG_LOG_HOME=<<Log_File_Path>>
```

Before providing this specification, make sure you create a folder for BTM logs - say, **eGBTMLogs** - in any directory to which the target application server has access. Then, against, -DEG_LOG_HOME, provide the full path to the **eGBTMLogs** directory. Where multiple instances on the same server are to be BTM-enabled, you can use the same directory for writing log files of all instances.

For example, to create log files in the **/App001/eGBTMLogs** directory, the complete specification will be as follows:

```
-DEG_PROPS_HOME=/opt/btm/AppServer1_8088
```

```
-DEG_LOG_HOME=/App001/eGBTMLogs
```

```
"-javaagent:/opt/btm/AppServer1_8088/eg_btm.jar”
```

20. Finally, save the file, and restart the JVM node.
21. If multiple JVM instances on a single node are monitored, make sure you follow steps 17-20 for each JVM instance.

2.2 Installing eG Java BTM on an Apache Tomcat Server

The steps for deploying the eG Java BTM on a Tomcat server will differ based on the platform on which the target Tomcat server is running - whether on a Windows platform or a Unix platform.

2.2.1 BTM-Enabling a Tomcat Server Running on a Windows Platform

If the Tomcat server is running on a Windows operating system, then follow the steps below to BTM-enable that server:

1. Login to the Tomcat server.
2. Open a browser on the server, connect to the eG manager, and login to the eG admin interface.
3. Manage the Tomcat server as a separate component using the eG administrative interface. When managing, make a note of the **Nick name** and **Port number** that you provide.
4. If multiple Tomcat server instances are operating on a single host, and you want to BTM-enable all the instances, then you will have to manage each instance as a separate component using the eG administrative interface. When doing so, make a note of the **Nick name** and **Port number** using which you managed each instance.
5. Next, log out of the eG admin interface. Then, create a **btm** directory anywhere on the Tomcat server - say, **C:\btm**. Under this directory, create a sub-folder. Make sure that you name this sub-

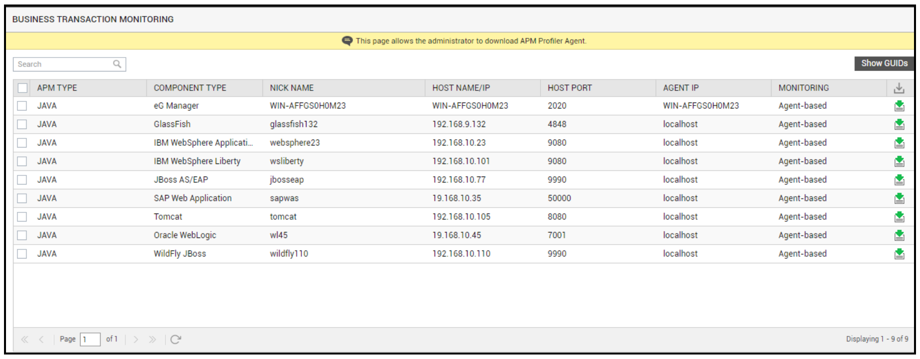
folder in the following format: <*Managed_Component_NickName*>_<*Managed_Component_Port*>. For instance, if you have managed the Tomcat server using the nick name *tomcat1* and the port number *8080*, the new directory under the **btm** directory should be named as *tomcat1_8080*.

6. If you have managed multiple Tomcat server instances running on a single host, then you will have to create multiple sub-directories under the **btm** directory- one each for every instance. Each of these sub-directories should be named after the **Nick name** and **Port number** using which the corresponding instance has been managed in eG.

7. Once the new sub-directory(ies) is created, open a browser on the Tomcat server, connect to the eG manager, and login to the eG admin interface again.

8. Follow the Agents -> BTM Profiler Download menu sequence in the eG admin interface.

9. Figure 2.3 will appear listing the servers that can be instrumented for APM by eG. In this list, locate the Tomcat server that you want to BTM-enable. Once you locate the server, click the **Download** icon corresponding to that server to download the **APM Profiler Agent** to it. If multiple Tomcat server instances on a single host are managed, then you will have to download the APM Profiler Agent separately for each of the managed instances.



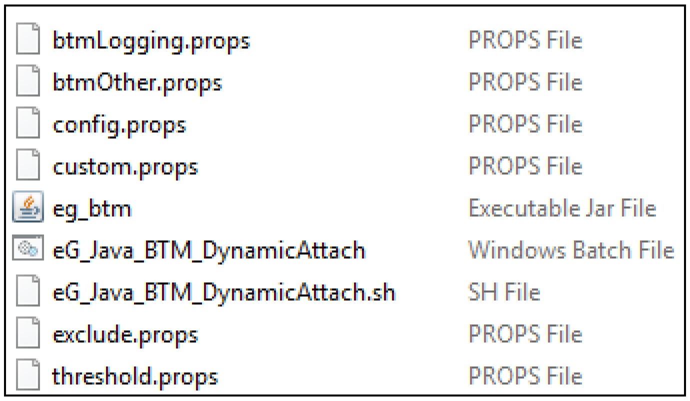
This page allows the administrator to download APM Profiler Agent.

APM TYPE	COMPONENT TYPE	NICK NAME	HOST NAME/IP	HOST PORT	AGENT IP	MONITORING
JAVA	eG Manager	WIN-AFFGS0H0M23	WIN-AFFGS0H0M23	2020	WIN-AFFGS0H0M23	Agent-based
JAVA	GlassFish	glassfish132	192.168.9.132	4848	localhost	Agent-based
JAVA	IBM WebSphere Applicati...	websphere23	192.168.10.23	9080	localhost	Agent-based
JAVA	IBM WebSphere Liberty	wsliberty	192.168.10.101	9080	localhost	Agent-based
JAVA	JBoss AS/EAP	jbosseap	192.168.10.77	9990	localhost	Agent-based
JAVA	SAP Web Application	sapwas	19.168.10.35	50000	localhost	Agent-based
JAVA	Tomcat	tomcat	192.168.10.105	8080	localhost	Agent-based
JAVA	Oracle WebLogic	wl45	19.168.10.45	7001	localhost	Agent-based
JAVA	WildFly JBoss	wildfly110	192.168.10.110	9990	localhost	Agent-based

Figure 2.3: Downloading the APM Profiler Agent for the Tomcat server

10. Upon clicking the **Download** icon in Figure 2.3, a zip file named **javaagent_<Nick_name_of_Tomcat_server>_<Port_number_of_Tomcat_server** will get downloaded. For instance, if you have managed the Tomcat server using the nickname 'tomcat1' and the port number '8080', then the name of the zip file will be *javaagent_tomcat1_8080*. Where multiple Tomcat server instances have been managed, you will be downloading multiple zip files - one each for every instance. The names of these zip files will automatically carry the nick name and port number you assigned to the corresponding server instance.

11. Copy the downloaded zip file(s) to the corresponding sub-directory(ies) of the **btm** directory (see steps 5 and 6 above). For example, the zip file named *javaagent_tomcat1_8080*, should be copied to the **C:\btm\tomcat1_8080** directory .
12. Extract the contents of each zip file into the same sub-directory to which that zip file was copied.
13. Figure 2.4 depicts the extracted contents of the zip file.



btmLogging.props	PROPS File
btmOther.props	PROPS File
config.props	PROPS File
custom.props	PROPS File
eg_btm	Executable Jar File
eG_Java_BTM_DynamicAttach	Windows Batch File
eG_Java_BTM_DynamicAttach.sh	SH File
exclude.props	PROPS File
threshold.props	PROPS File

Figure 2.4: Contents of the APM Profiler Agent zip

14. From Figure 2.4, it is evident that the zip file contains an **eg_btm.jar** file and a few property files, namely - **btmOther.props** , **btmLogging.props** , **config.props** , **custom.props** , **exclude.props**, and **threshold.props** files.
15. Next, proceed to BTM-enable the Tomcat server/instance. For that, edit the **btmOther.props** file in the sub-directory (of the **btm** directory) that corresponds to that Tomcat server/ instance. You will find the following lines in the file:

```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Below property is BTM Server Socket Port, through which eG Agent Communicates
# Restart is required, if any changes in this property
# Default port is "13931"
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#
BTM_Port=13931
#
```

By default, the **BTMPort** parameter is set to 13931. If you want to enable eG Java BTM on a different port, then specify the same here. In this case, when configuring the **Java Business**

Transactions test or the **Key Java Business Transactions** test for that application server, make sure you configure the **BTM** port parameter of the test with this port number.

Note:

When BTM-enabling multiple instances on the same server, make sure you configure a different **BTM Port** for each instance.

Also, by default, the **Designated_Agent** parameter will be empty; do not disturb this default setting. In this case therefore, the eG Java BTM will treat the host from which the very first 'measure request' comes in as the **Designated_Agent**.

Note:

In case a specific **Designated_Agent** is not provided, and the eG Java BTM treats the host from which the very first 'measure request' comes in as the **Designated_Agent**, then if such a **Designated_Agent** is stopped or uninstalled for any reason, the eG Java BTM will wait for a maximum of 10 measure periods for that 'deemed' **Designated_Agent** to request for metrics. If no requests come in for 10 consecutive measure periods, then the eG Java BTM will begin responding to 'measure requests' coming in from any other eG agent.

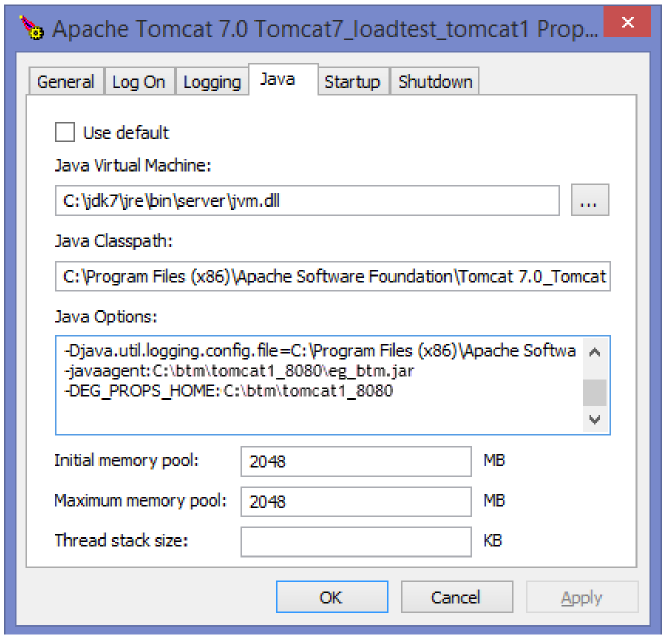
16. Finally, save the **btmOther.props** file.

17. Then, you need to configure the Tomcat server/server instance with the path to the **eg_btm.jar** and **.props** files. This can be done, in one of the following ways:

    - Through the Tomcat control panel;
    - Through the Tomcat start-up script

18. To use the control panel, do the following:

- First, open the Tomcat Control Panel.



Use default

Java Virtual Machine: C:\jdk7\jre\bin\server\jvm.dll

Java Classpath: C:\Program Files (x86)\Apache Software Foundation\Tomcat 7.0_Tomcat

Java Options:

```
-Djava.util.logging.config.file=C:\Program Files (x86)\Apache Softwa
-javaagent:C:\btm\tomcat1_8080\eg_btm.jar
-DEG_PROPS_HOME:C:\btm\tomcat1_8080
```

Initial memory pool: 2048 MB

Maximum memory pool: 2048 MB

Thread stack size: KB

Figure 2.5: BTM-enabling the Tomcat server on Windows

- Select the **Java** tab page in Section **2.2** above.
- Add entries of the following format to the **Java Options** section of 2.2:

```
-javaagent:<<PATH_TO_THE_eg_btm.jar_FILE>>
-DEG_PROPS_HOME=<<PATH_TO_THE_LOCAL_FOLDER_CONTAINING_THE_PROPS_FILES>>
```

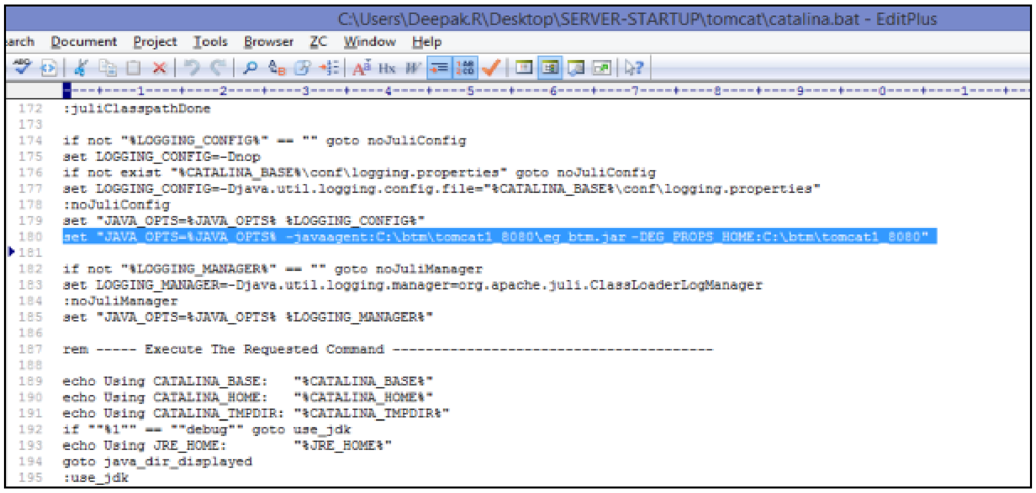
  For instance, if the .props files and the eg_btm.jar had been copied to **C:\btm\tomcat1_8080**, the above specification will be:

```
-javaagent:C:\btm\tomcat1_8080\eg_btm.jar
-DEG_PROPS_HOME=C:\btm\tomcat1_8080
```

- Click the **Apply** and **OK** buttons in 2.2.
- Restart the Tomcat service.
- Where multiple Tomcat server instances on a host are to be monitored, repeat steps 15, 16, and 18 for each of the server instances.

19. On the other hand, if you want to configure using the Tomcat start-up script, follow the steps below:

- Open the **catalina.bat** file from the <TOMCAT_HOME> directory on the Tomcat server.
- Insert the lines of code indicated by 2.2 to BTM-enable the Tomcat server.


C:\Users\Deepak.R\Desktop\SERVER-STARTUP\tomcat\catalina.bat - EditPlus
arch Document Project Tools Browser ZC Window Help

```
:juliClasspathDone

if not "%LOGGING_CONFIG%" == "" goto noJuliConfig
set LOGGING_CONFIG=-Dnop
if not exist "%CATALINA_BASE%\conf\logging.properties" goto noJuliConfig
set LOGGING_CONFIG=-Djava.util.logging.config.file="%CATALINA_BASE%\conf\logging.properties"
:noJuliConfig
set "JAVA_OPTS=%JAVA_OPTS% %LOGGING_CONFIG%"
set "JAVA_OPTS=%JAVA_OPTS% -javaagent:C:\btm\tomcat1_8080\eg_btm.jar -DEG_PROPS_HOME:C:\btm\tomcat1_8080"

if not "%LOGGING_MANAGER%" == "" goto noJuliManager
set LOGGING_MANAGER=-Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager
:noJuliManager
set "JAVA_OPTS=%JAVA_OPTS% %LOGGING_MANAGER%"

rem ----- Execute The Requested Command ---------------------------------------

echo Using CATALINA_BASE:   "%CATALINA_BASE%"
echo Using CATALINA_HOME:   "%CATALINA_HOME%"
echo Using CATALINA_TMPDIR: "%CATALINA_TMPDIR%"
if ""%1"" == ""debug"" goto use_jdk
echo Using JRE_HOME:        "%JRE_HOME%"
goto java_dir_displayed
:use_jdk
```

Figure 2.6: Editing the catalina.bat file

- Save the file and restart the Tomcat server.
- Where multiple Tomcat server instances on a host are to be monitored, repeat steps 15, 16, and 19 for each of the server instances.

2.2.2 BTM-Enabling a Tomcat Server Running on a Unix Platform

If the target Tomcat server is running on a Unix operating system, then follow the steps below to BTM-enable that server:

1. Login to any system in your environment that supports a browser and has network access to the eG manager.
2. Open a browser on that system, connect to the eG manager, and login to the eG admin interface.
3. Manage the target Tomcat server as a separate component using the eG administrative interface. When managing, make a note of the **Nick name** and **Port number** that you provide.
4. If multiple Tomcat server instances are operating on a single host, and you want to monitor each of those instances, then you will have to manage each instance as a separate Tomcat server component using the eG administrative interface. When doing so, make a note of the **Nick name** and **Port number** using which you managed each instance.
5. Next, log out of the eG admin interface and the system.

6. Log into the target Tomcat server. Then, create a **btm** directory anywhere on the target server - say, **/opt/btm**. Under this directory, create a sub-folder. Make sure that you name this sub-folder in the following format: *<Managed_Component_NickName>_<Managed_Component_Port>*. For instance, if you have managed the Tomcat server using the nick name *tomcat1* and the port number *8080*, the new directory under the **btm** directory should be named as *tomcat1_8080*.
7. If you have managed multiple Tomcat server instances running on a single host, then you will have to create multiple sub-directories under the **btm** directory- one each for every instance. Each of these sub-directories should be named after the **Nick name** and **Port number** using which the corresponding instance has been managed in eG.
8. Once the new sub-directory(ies) is created, log out of the Tomcat server. Log back into the system you used in step 1 above. Open a browser on the system, connect to the eG manager, and login to the eG admin interface again.
9. Follow the Agents -> BTM Profiler Download menu sequence in the eG admin interface.
10. Figure 2.3 will appear listing the servers that can be instrumented for APM by eG. In this list, locate the Tomcat server that you want to BTM-enable. Once you locate the server, click the **Download** icon corresponding to that server to download the **APM Profiler Agent** to that server. If multiple Tomcat server instances on a single host are managed, then you will have to download the APM Profiler Agent separately for each of the managed instances.
11. Upon clicking the **Download** icon in Figure 2.3, a zip file named **javaagent_<Nick_name_of_ Tomcat_server>_<Port_number_of_Tomcat_server** will get downloaded. For instance, if you have managed the Tomcat server using the nickname 'tomcat1' and the port number '8080', then the name of the zip file will be *javaagent_tomcat1_8080*. Where multiple Tomcat server instances have been managed, you will be downloading multiple zip files - one each for every instance. The names of these zip files will automatically carry the nick name and port number you assigned to the corresponding Tomcat server instance.
12. Next, using FTP tools like putty or WinSCP, transfer the downloaded zip file(s) to the corresponding sub-directory(ies) (of the **btm** directory), which you created on the Tomcat server at steps 6 and 7 above. For example, the zip file named *javaagent_tomcat1_8080*, should be transferred to the**/opt/btm/tomcat1_8080** directory on the target Tomcat server.
13. Log out of the system and log back into the Tomcat server.
14. Extract the contents of each zip file into the same sub-directory to which that zip file was copied.
15. Figure 2.4 depicts the extracted contents of the zip file.
16. From Figure 2.4, it is evident that the zip file contains an **eg_btm.jar** file and a few property files,

namely - **btmOther.props** , **btmLogging.props** , **config.props** , **custom.props** , **exclude.props**, and **threshold.props** files.

17. Next, proceed to BTM-enable the Tomcat server/instance. For that, edit the **btmOther.props** file in the sub-directory (of the **btm** directory) that corresponds to that server/ instance. You will find the following lines in the file:

```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Below property is BTM Server Socket Port, through which eG Agent Communicates
# Restart is required, if any changes in this property
# Default port is "13931"
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#
BTM_Port=13931
#
```

By default, the **BTMPort** parameter is set to 13931. If you want to enable eG Java BTM on a different port, then specify the same here. In this case, when configuring the **Java Business Transactions** test or the **Key Java Business Transactions** test for that application server, make sure you configure the **BTM PORT** parameter of the test with this port number.

Note:

When BTM-enabling multiple instances on the same server, make sure you configure a different **BTM Port** for each instance.

Also, by default, the **Designated_Agent** parameter will be empty; do not disturb this default setting. In this case therefore, the eG Java BTM will treat the host from which the very first 'measure request' comes in as the **Designated_Agent**.

Note:

In case a specific **Designated_Agent** is not provided, and the eG Java BTM treats the host from which the very first 'measure request' comes in as the **Designated_Agent**, then if such a **Designated_Agent** is stopped or uninstalled for any reason, the eG Java BTM will wait for a maximum of 10 measure periods for that 'deemed' **Designated_Agent** to request for metrics. If no requests come in for 10 consecutive measure periods, then the eG Java BTM will begin responding to 'measure requests' coming in from any other eG agent.

18. Finally, save the **btmOther.props** file.

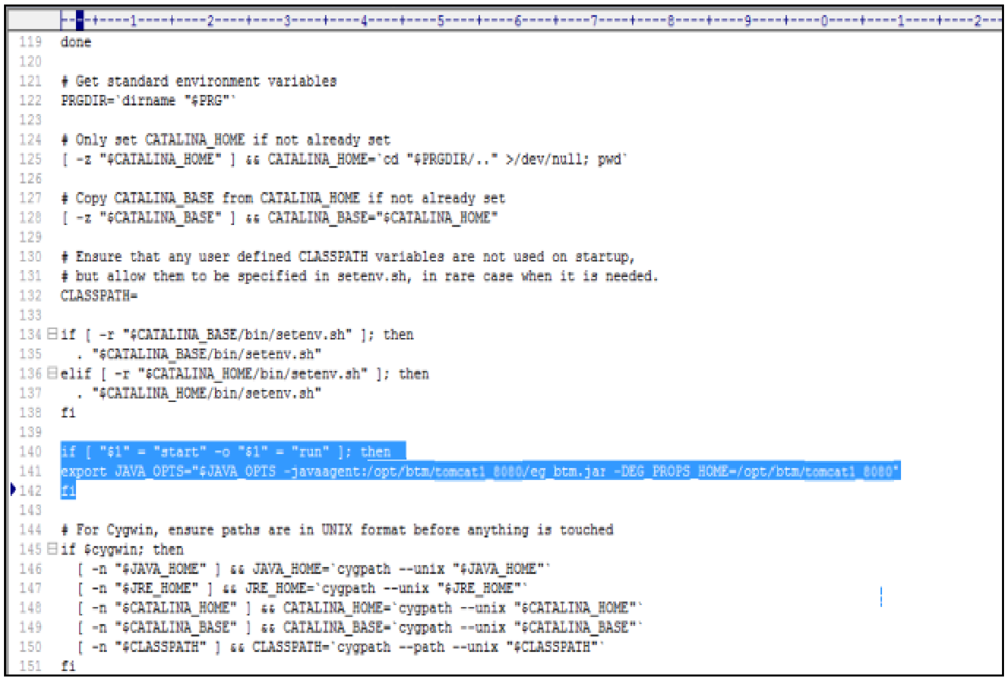
19. Then, you need to configure the Tomcat server with the path to the **eg_btm.jar** and **.props** files. This can be done by editing the start-up script of the Tomcat server. For that, first open the start-up script.

20. Insert the following lines in the script (as depicted by Figure 2.7) to BTM-enable the server.

```
if [ "$1" = "start" -o "$1" = "run" ]; then
export  JAVA_ OPTS="$JAVA_ OPTS  - javaagent:<<PATH  TO  THE  eg_ btm.jar>>  - DEG_ PROPS_
HOME=<<PATH TO LOCAL FOLDER CONTAINING THE
.PROPS FILES>>
fi
```

For instance, if the eg_btm.jar file and .props files have been copied to the **tomcat1_8080** folder within the **/opt//btm** folder, then your specification will be as follows:

```
if [ "$1" = "start" -o "$1" = "run" ]; then
export JAVA_OPTS="$JAVA_OPTS -javaagent:/opt/btm/tomcat1_8080/eg_btm.jar -DEG_PROPS_
HOME=/opt/btm/tomcat1_8080
fi
```



```
done

# Get standard environment variables
PRGDIR=`dirname "$PRG"`

# Only set CATALINA_HOME if not already set
[ -z "$CATALINA_HOME" ] && CATALINA_HOME=`cd "$PRGDIR/.." >/dev/null; pwd`

# Copy CATALINA_BASE from CATALINA_HOME if not already set
[ -z "$CATALINA_BASE" ] && CATALINA_BASE="$CATALINA_HOME"

# Ensure that any user defined CLASSPATH variables are not used on startup,
# but allow them to be specified in setenv.sh, in rare case when it is needed.
CLASSPATH=

if [ -r "$CATALINA_BASE/bin/setenv.sh" ]; then
  . "$CATALINA_BASE/bin/setenv.sh"
elif [ -r "$CATALINA_HOME/bin/setenv.sh" ]; then
  . "$CATALINA_HOME/bin/setenv.sh"
fi

if [ "$1" = "start" -o "$1" = "run" ]; then
export JAVA_OPTS="$JAVA_OPTS -javaagent:/opt/btm/tomcat1_8080/eg_btm.jar -DEG_PROPS_HOME=/opt/btm/tomcat1_8080"
fi

# For Cygwin, ensure paths are in UNIX format before anything is touched
if $cygwin; then
  [ -n "$JAVA_HOME" ] && JAVA_HOME=`cygpath --unix "$JAVA_HOME"`
  [ -n "$JRE_HOME" ] && JRE_HOME=`cygpath --unix "$JRE_HOME"`
  [ -n "$CATALINA_HOME" ] && CATALINA_HOME=`cygpath --unix "$CATALINA_HOME"`
  [ -n "$CATALINA_BASE" ] && CATALINA_BASE=`cygpath --unix "$CATALINA_BASE"`
  [ -n "$CLASSPATH" ] && CLASSPATH=`cygpath --path --unix "$CLASSPATH"`
fi
```

Figure 2.7: Editing the start-up script of a Tomcat server on Linux to BTM-enable the server

21. In Unix environments, if the eG agent is deployed on the same host as the Tomcat server, then

both the agent and the server will be running using different user privileges. In this situation, by default, the eG Java BTM logs will not be created. In order to create the same, insert the following entry after the -DEG_PROPS_HOME specification and before the closing quotes .

```
-DEG_LOG_HOME=<LogFile_Path>
```

Before providing this specification, make sure you create a folder for BTM logs - say, **eGBTMLogs** - in any directory to which the target application server has access. Then, against, -DEG_LOG_HOME, provide the full path to the **eGBTMLogs** directory. Where multiple instances on the same server are to be BTM-enabled, you can use the same directory for writing log files of all instances.

For example, to create log files in the **/App001/eGBTMLogs** directory, the complete specification will be as follows:

```
if [ "$1" = "start" -o "$1" = "run" ]; then
```

```
export JAVA_OPTS="$JAVA_OPTS -javaagent:/opt/btm/tomcat1_8080/eg_btm.jar -DEG_PROPS_
HOME=/opt/btm/tomcat1_8080 -DEG_LOG_HOME=/App001/eGBTMLogs
```

```
fi
```

22. Finally, save the file and restart the Tomcat server.
23. Where multiple Tomcat server instances on a host are to be monitored, repeat steps 17 to 22 for each of the server instances.

2.3 Installing eG Java BTM on an IBM WebSphere

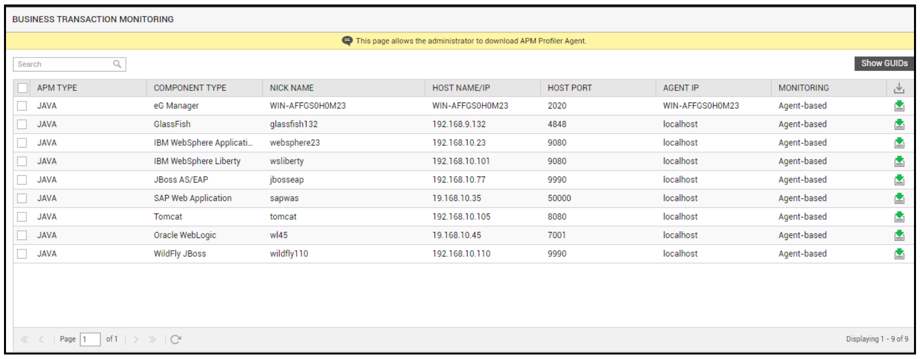
The steps for deploying the eG Java BTM on a WebSphere server will differ based on the platform on which the target WebSphere server is running - whether on a Windows platform or a Unix platform.

2.3.1 BTM-Enabling a WebSphere Server Running on a Windows Platform

If the WebSphere server is running on a Windows operating system, then follow the steps below to BTM-enable that server:

1. Login to the WebSphere server.
2. Open a browser on the server, connect to the eG manager, and login to the eG admin interface.

3. Manage the WebSphere server as a separate component using the eG administrative interface. When managing, make a note of the **Nick name** and **Port number** that you provide.
4. If multiple WebSphere server instances are operating on a single host, and you want to BTM-enable all the instances, then you will have to manage each instance as a separate component using the eG administrative interface. When doing so, make a note of the **Nick name** and **Port number** using which you managed each instance.
5. Next, log out of the eG admin interface. Then, create a **btm** directory anywhere on the WebSphere server - say, **C:\btm**. Under this directory, create a sub-folder. Make sure that you name this sub-folder in the following format: <*Managed_Component_NickName*>_<*Managed_Component_Port*>. For instance, if you have managed the WebSphere server using the nick name *WebSphere1* and the port number *9080*, the new directory under the **btm** directory should be named as *WebSphere1_9080*.
6. If you have managed multiple WebSphere server instances running on a single host, then you will have to create multiple sub-directories under the **btm** directory- one each for every instance. Each of these sub-directories should be named after the **Nick name** and **Port number** using which the corresponding instance has been managed in eG.
7. Once the new sub-directory(ies) is created, open a browser on the WebSphere server, connect to the eG manager, and login to the eG admin interface again.
8. Follow the Agents -> BTM Profiler Download menu sequence in the eG admin interface.
9. Figure 2.8 will appear listing the servers that can be instrumented for APM by eG. In this list, locate the WebSphere server that you want to BTM-enable. Once you locate the server, click the **Download** icon corresponding to that server to download the **APM Profiler Agent** to that server. If multiple WebSphere server instances on a single host are managed, then you will have to download the APM Profiler Agent separately for each of the managed instances.


BUSINESS TRANSACTION MONITORING

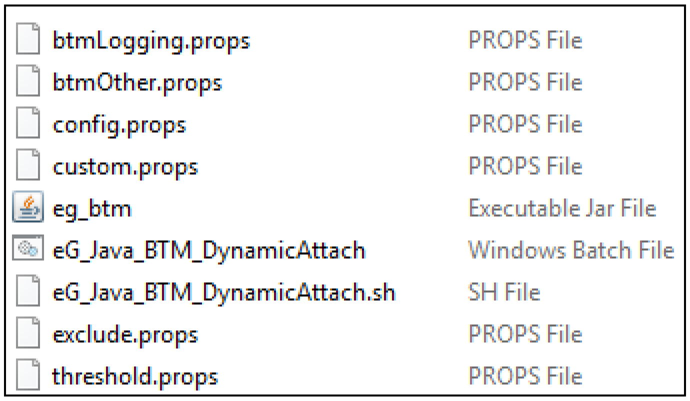
This page allows the administrator to download APM Profiler Agent.

APM TYPE	COMPONENT TYPE	NICK NAME	HOST NAME/IP	HOST PORT	AGENT IP	MONITORING
JAVA	eG Manager	WIN-AFFGS0H0M23	WIN-AFFGS0H0M23	2020	WIN-AFFGS0H0M23	Agent-based
JAVA	GlassFish	glassfish132	192.168.9.132	4848	localhost	Agent-based
JAVA	IBM WebSphere Applicati...	websphere23	192.168.10.23	9080	localhost	Agent-based
JAVA	IBM WebSphere Liberty	wsliberty	192.168.10.101	9080	localhost	Agent-based
JAVA	JBoss AS/EAP	jbosseap	192.168.10.77	9990	localhost	Agent-based
JAVA	SAP Web Application	sapwas	19.168.10.35	50000	localhost	Agent-based
JAVA	Tomcat	tomcat	192.168.10.105	8080	localhost	Agent-based
JAVA	Oracle WebLogic	wl45	19.168.10.45	7001	localhost	Agent-based
JAVA	WildFly JBoss	wildfly110	192.168.10.110	9990	localhost	Agent-based

Displaying 1 - 9 of 9

Figure 2.8: Downloading the APM Profiler Agent for the WebSphere server

10. Upon clicking the **Download** icon in Figure 2.8, a zip file named **javaagent_<Nick_name_of_ WebSphere_server>_<Port_number_of_WebSphere_server** will get downloaded. For instance, if you have managed the WebSphere server using the nickname 'WebSphere1' and the port number '9080', then the name of the zip file will be *javaagent_WebSphere1_9080*. Where multiple WebSphere server instances have been managed, you will be downloading multiple zip files - one each for every instance. The names of these zip files will automatically carry the nick name and port number you assigned to the corresponding server instance.
11. Copy the downloaded zip file(s) to the corresponding sub-directory(ies) of the **btm** directory (see steps 5 and 6 above). For example, the zip file named *javaagent_WebSphere1_9080*, should be copied to the **C:\btm\WebSphere1_9080** directory .
12. Extract the contents of each zip file into the same sub-directory to which that zip file was copied.
13. Figure 2.9 depicts the extracted contents of the zip file.



btmLogging.props	PROPS File
btmOther.props	PROPS File
config.props	PROPS File
custom.props	PROPS File
eg_btm	Executable Jar File
eG_Java_BTM_DynamicAttach	Windows Batch File
eG_Java_BTM_DynamicAttach.sh	SH File
exclude.props	PROPS File
threshold.props	PROPS File

Figure 2.9: Contents of the APM Profiler Agent zip

14. From Figure 2.9, it is evident that the zip file contains an **eg_btm.jar** file and a few property files, namely - **btmOther.props** , **btmLogging.props** , **config.props** , **custom.props** , **exclude.props**, and **threshold.props** files.
15. Next, proceed to BTM- enable the WebSphere server/instance. For that, edit the **btmOther.props** file in the sub- directory (of the **btm** directory) that corresponds to that WebSphere server/ instance. You will find the following lines in the file:

```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Below property is BTM Server Socket Port, through which eG Agent Communicates
# Restart is required, if any changes in this property
# Default port is "13931"
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
#
BTM_Port=13931
#
```

By default, the **BTMPort** parameter is set to 13931. If you want to enable eG Java BTM on a different port, then specify the same here. In this case, when configuring the **Java Business Transactions** test or the **Key Java Business Transactions** test for that application server, make sure you configure the **BTM** port parameter of the test with this port number.

Note:

When BTM-enabling multiple instances on the same server, make sure you configure a different **BTM Port** for each instance.

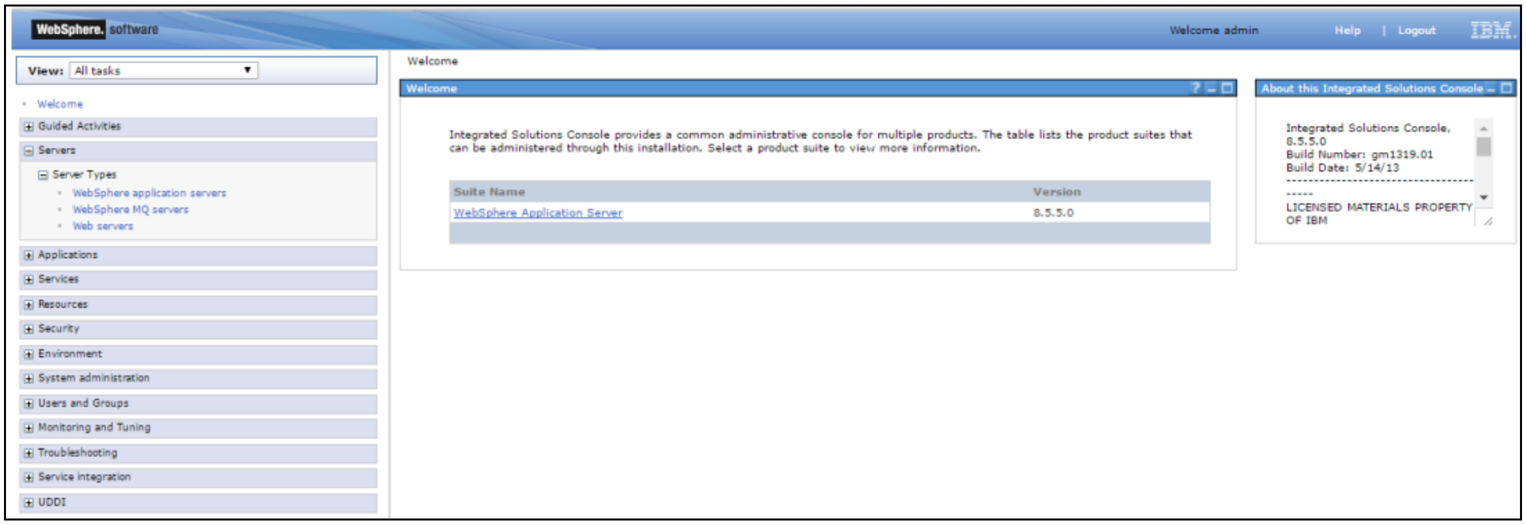
Also, by default, the **Designated_Agent** parameter will be empty; do not disturb this default setting. In this case therefore, the eG Java BTM will treat the host from which the very first 'measure request' comes in as the **Designated_Agent**.

Note:

In case a specific **Designated_Agent** is not provided, and the eG Java BTM treats the host from which the very first 'measure request' comes in as the **Designated_Agent**, then if such a **Designated_Agent** is stopped or uninstalled for any reason, the eG Java BTM will wait for a maximum of 10 measure periods for that 'deemed' **Designated_Agent** to request for metrics. If no requests come in for 10 consecutive measure periods, then the eG Java BTM will begin responding to 'measure requests' coming in from any other eG agent.

16. Finally, save the **btmOther.props** file.

17. Then, you need to configure the WebSphere server with the path to the **eg_btm.jar** and **.props** files. For this, first login to the WebSphere administration console. When Figure 2.10 appears, click on the **WebSphere Application Server** link in the right panel.



Integrated Solutions Console provides a common administrative console for multiple products. The table lists the product suites that can be administered through this installation. Select a product suite to view more information.

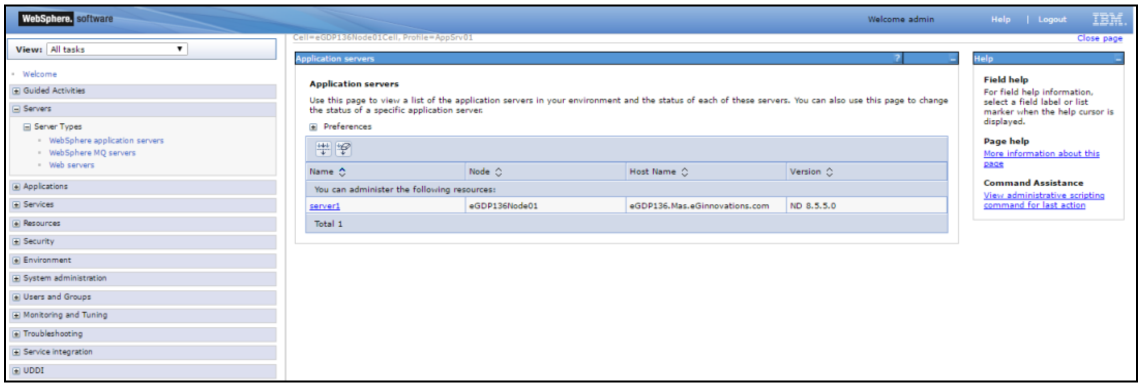
Suite Name	Version
WebSphere Application Server	8.5.5.0

Integrated Solutions Console, 8.5.5.0
Build Number: gm1319.01
Build Date: 5/14/13

LICENSED MATERIALS PROPERTY OF IBM

Figure 2.10: The WebSphere Administration console

18. This will invoke Figure 2.11. In the right panel of Figure 2.11, click on the link representing the WebSphere server instance that you want to BTM-enable.



Cell=eGDP136Node01Cell, Profile=AppSrv01

Application servers

Use this page to view a list of the application servers in your environment and the status of each of these servers. You can also use this page to change the status of a specific application server.

Preferences

Name	Node	Host Name	Version
You can administer the following resources:			
server1	eGDP136Node01	eGDP136.Mas.eGinnovations.com	ND 8.5.5.0
Total 1			

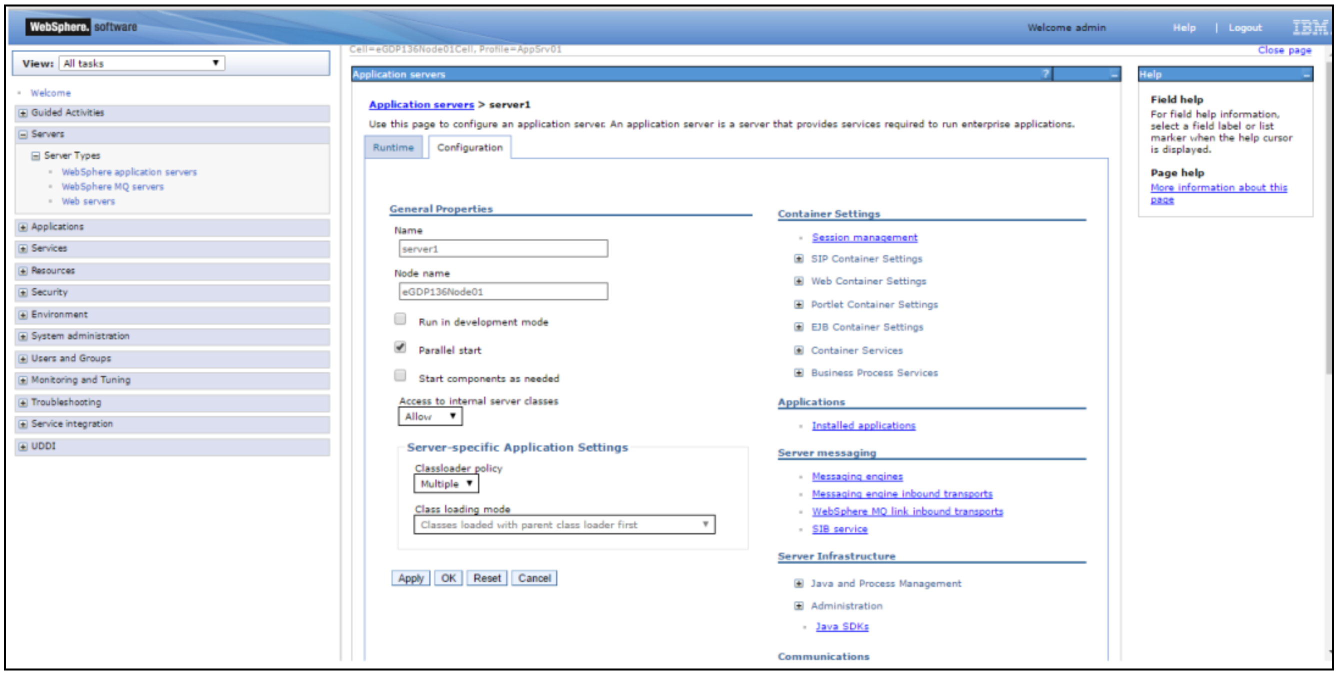
Field help
For field help information, select a field label or list marker when the help cursor is displayed.

Page help
More information about this page

Command Assistance
View administrative scripting command for last action

Figure 2.11: Clicking on the WebSphere server instance to be BTM-enabled

19. Figure 2.12 will then appear.



Cell=eGDP136Node01Cell, Profile=AppSrv01

Application servers > server1

Use this page to configure an application server. An application server is a server that provides services required to run enterprise applications.

General Properties

Name: server1

Node name: eGDP136Node01

☐ Run in development mode

☑ Parallel start

☐ Start components as needed

Access to internal server classes: Allow

Server-specific Application Settings

Classloader policy: Multiple

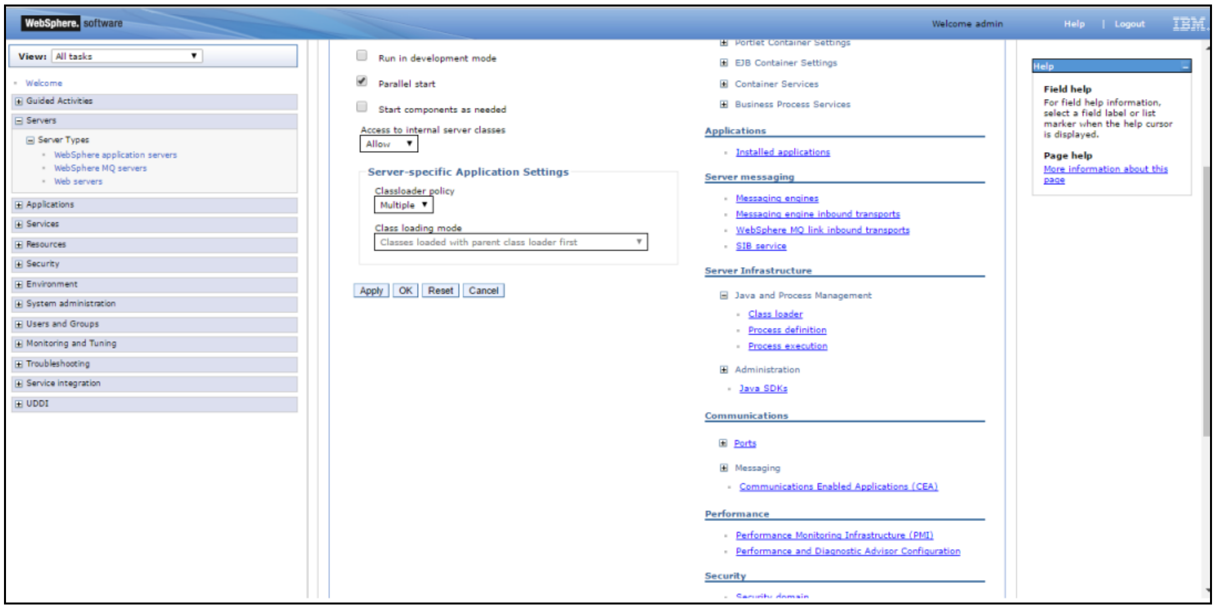
Class loading mode: Classes loaded with parent class loader first

Field help
For field help information, select a field label or list marker when the help cursor is displayed.

Page help
More information about this page

Figure 2.12: The Configuration tab page of the WebSphere server instance to be BTM-enabled

20. Keep scrolling down the right panel of Figure 2.13 until you find the **Server Infrastructure** section. Expand the **Java and Process Management** node in that section, and click on the **Process definition** link within.



☐ Run in development mode

☑ Parallel start

☐ Start components as needed

Access to internal server classes: Allow

Server-specific Application Settings

Classloader policy: Multiple

Class loading mode: Classes loaded with parent class loader first

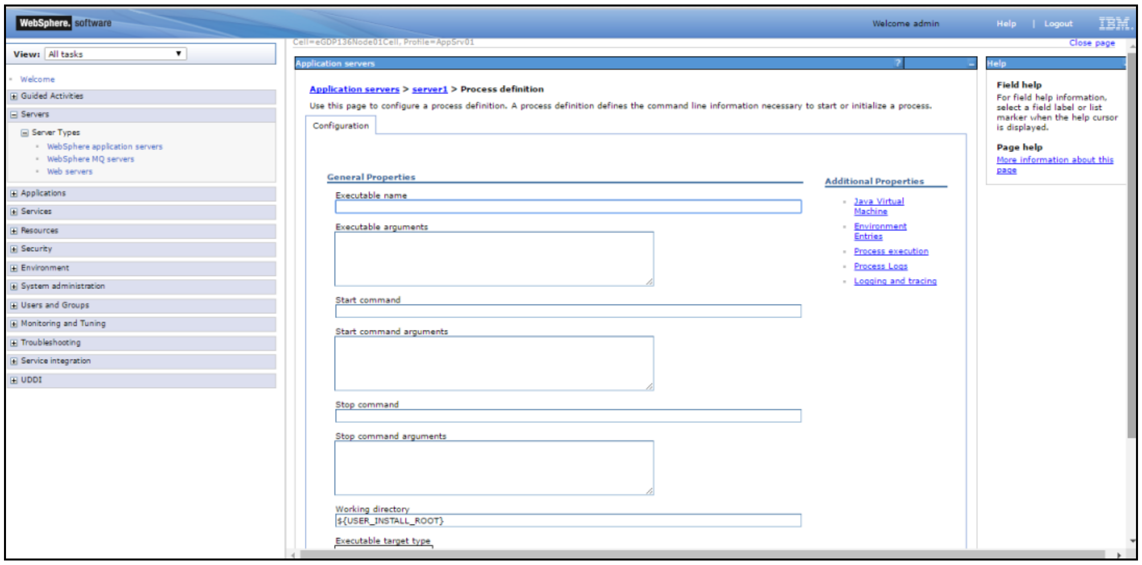
Field help
For field help information, select a field label or list marker when the help cursor is displayed.

Page help
More information about this page

Figure 2.13: Selecting the Process definition option from Java and Process Management tree

21. Figure 2.14 will then appear. From the **Additional Properties** section, select **Java Virtual**

Machines.



Cell=eGDP136Node01Cell, Profile=AppSrv01

Application servers

Application servers > server1 > Process definition

Use this page to configure a process definition. A process definition defines the command line information necessary to start or initialize a process.

General Properties

Executable name

Executable arguments

Start command

Start command arguments

Stop command

Stop command arguments

Working directory: ${USER_INSTALL_ROOT}

Executable target type

Additional Properties

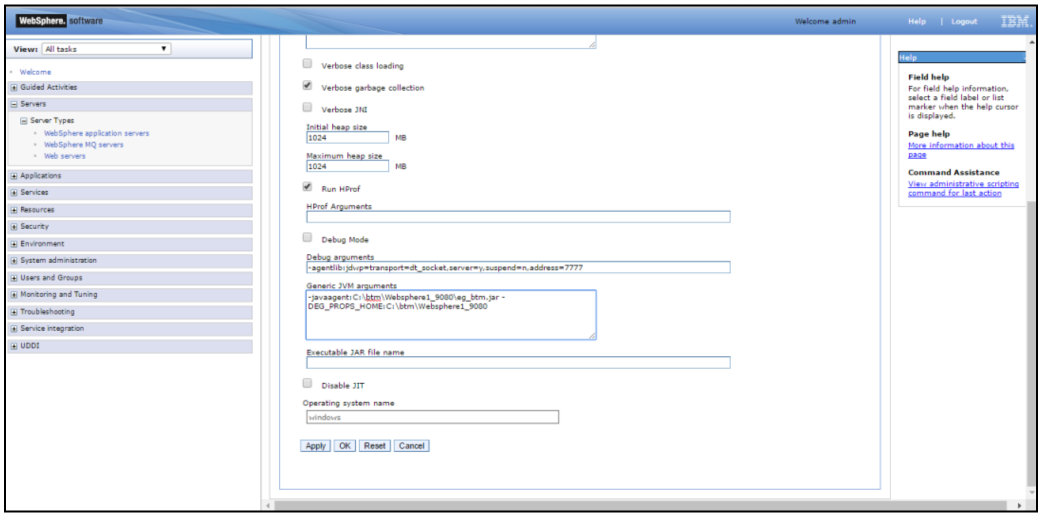
Help

Field help
For field help information, select a field label or list marker when the help cursor is displayed.

Page help
More information about this page

Figure 2.14: Configuring the Process definition

22. When Figure 2.15 appears, scroll down its right panel until the **Generic JVM Arguments** text box comes into view.



☐ Verbose class loading

☑ Verbose garbage collection

☐ Verbose JNI

Initial heap size: 1024 MB

Maximum heap size: 1024 MB

☑ Run HProf

HProf Arguments

☐ Debug Mode

Debug arguments: -agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=7777

Generic JVM arguments:
-javaagent:C:\btm\Websphere1_9080\eg_btm.jar -
DEG_PROPS_HOME:C:\btm\Websphere1_9080

Executable JAR file name

☐ Disable JIT

Operating system name: windows

Help

Field help
For field help information, select a field label or list marker when the help cursor is displayed.

Page help
More information about this page

Command Assistance
View administrative scripting command for last action

Figure 2.15: Configuring the JVM arguments for a WebSphere Server on Windows

23. Here, specify the following:

```
-javaagent:<<PATH TO THE eg_btm.jar FILE>>
```

```
-DEG_PROPS_HOME=<<PATH OF THE LOCAL FOLDER CONTAINING THE .PROPS FILES>>
```

For instance, if the eg_ btm.jar file and .props files have been copied to the **C:\btm\Websphere1_9080** directory, the above specification will be:

```
-javaagent:C:\btm\WebSphere1_9080\eg_btm.jar
```

```
-DEG_PROPS_HOME=C:\btm\Websphere1_9080
```

24. Save the changes and restart the WebSphere server.
25. Where multiple instances of WebSphere are monitored, make sure you perform step 15- 24 above for each WebSphere server instance.

2.3.2 BTM-Enabling a WebSphere Server Running on a Unix Platform

If the target WebSphere server is running on a Unix operating system, then follow the steps below to BTM-enable that server:

1. Login to any system in your environment that supports a browser and has network access to the eG manager.
2. Open a browser on that system, connect to the eG manager, and login to the eG admin interface.
3. Manage the target WebSphere server as a separate component using the eG administrative interface. When managing, make a note of the **Nick name** and **Port number** that you provide.
4. If multiple WebSphere server instances are operating on a single host, and you want to monitor each of those instances, then you will have to manage each instance as a separate WebSphere server component using the eG administrative interface. When doing so, make a note of the **Nick name** and **Port number** using which you managed each instance.
5. Next, log out of the eG admin interface and the system.
6. Log into the target WebSphere server. Then, create a **btm** directory anywhere on the target server - say, **/opt/btm**. Under this directory, create a sub-folder. Make sure that you name this sub- folder in the following format: *<Managed_ Component_ NickName>_ <Managed_ Component_Port>*. For instance, if you have managed the WebSphere server using the nick name *tomcat1* and the port number *8080*, the new directory under the **btm** directory should be named as *tomcat1_8080*.
7. If you have managed multiple WebSphere server instances running on a single host, then you will have to create multiple sub-directories under the **btm** directory- one each for every instance.

Each of these sub-directories should be named after the **Nick name** and **Port number** using which the corresponding instance has been managed in eG.

8. Once the new sub-directory(ies) is created, log out of the WebSphere server. Log back into the system you used in step 1 above. Open a browser on the system, connect to the eG manager, and login to the eG admin interface again.
9. Follow the Agents -> BTM Profiler Download menu sequence in the eG admin interface.
10. Figure 2.8 will appear listing the servers that can be instrumented for APM by eG. In this list, locate the WebSphere server that you want to BTM-enable. Once you locate the server, click the **Download** icon corresponding to that server to download the **APM Profiler Agent** to that server. If multiple WebSphere server instances on a single host are managed, then you will have to download the APM Profiler Agent separately for each of the managed instances.
11. Upon clicking the **Download** icon in Figure 2.8, a zip file named **javaagent_<Nick_name_of_ WebSphere_ server>_<Port_ number_ of_ WebSphere_ server** will get downloaded. For instance, if you have managed the WebSphere server using the nickname 'WebSphere1' and the port number '9080', then the name of the zip file will be *javaagent_WebSphere1_9080*. Where multiple WebSphere server instances have been managed, you will be downloading multiple zip files - one each for every instance. The names of these zip files will automatically carry the nick name and port number you assigned to the corresponding WebSphere server instance.
12. Next, using FTP tools like putty or WinSCP, transfer the downloaded zip file (s) to the corresponding sub-directory(ies) (of the **btm** directory), which you created on the WebSphere server in steps 6 and 7 above. For example, the zip file named *javaagent_WebSphere1_9080*, should be transferred to the**/opt/btm/WebSphere1_9080** directory on the target WebSphere server.
13. Log out of the system and log back into the WebSphere server.
14. Extract the contents of each zip file into the same sub-directory to which that zip file was copied.
15. Figure 2.9 depicts the extracted contents of the zip file.
16. From Figure 2.9, it is evident that the zip file contains an **eg_btm.jar** file and a few property files, namely - **btmOther.props** , **btmLogging.props** , **config.props** , **custom.props** , **exclude.props**, and **threshold.props** files.
17. Next, proceed to BTM- enable the WebSphere server/instance. For that, edit the **btmOther.props** file in the sub-directory (of the **btm** directory) that corresponds to that server/ instance. You will find the following lines in the file:

```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Below property is BTM Server Socket Port, through which eG Agent Communicates
# Restart is required, if any changes in this property
# Default port is "13931"
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#
BTM_Port=13931
#
```

By default, the **BTMPort** parameter is set to 13931. If you want to enable eG Java BTM on a different port, then specify the same here. In this case, when configuring the **Java Business Transactions** test or the **Key Java Business Transactions** test for that application server, make sure you configure the **BTM PORT** parameter of the test with this port number.

Note:

When BTM-enabling multiple instances on the same server, make sure you configure a different **BTM Port** for each instance.

Also, by default, the **Designated_Agent** parameter will be empty; do not disturb this default setting. In this case therefore, the eG Java BTM will treat the host from which the very first 'measure request' comes in as the **Designated_Agent**.

Note:

In case a specific **Designated_Agent** is not provided, and the eG Java BTM treats the host from which the very first 'measure request' comes in as the **Designated_Agent**, then if such a **Designated_Agent** is stopped or uninstalled for any reason, the eG Java BTM will wait for a maximum of 10 measure periods for that 'deemed' **Designated_Agent** to request for metrics. If no requests come in for 10 consecutive measure periods, then the eG Java BTM will begin responding to 'measure requests' coming in from any other eG agent.

18. Finally, save the **btmOther.props** file.

19. Then, you need to configure the WebSphere server with the path to the **eg_btm.jar** and **.props** files. For this, first login to the WebSphere administration console. When Figure 2.10 appears, click on the **WebSphere Application Server** link in the right panel.

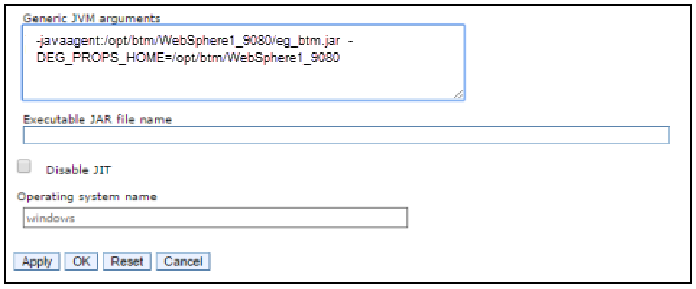
20. This will invoke Figure 2.11. In the right panel of Figure 2.11, click on the link representing the WebSphere server instance that you want to BTM-enable.

21. Figure 2.12 will then appear.

22. Keep scrolling down the right panel of Figure 2.13 until you find the **Server Infrastructure** section. Expand the **Java and Process Management** node in that section, and click on the **Process definition** link within.

23. Figure 2.14 will then appear. From the **Additional Properties** section, select **Java Virtual Machines**.

24. When Figure 2.15 appears, scroll down its right panel until the **Generic JVM Arguments** text box of Figure 2.15 comes into view.



Generic JVM arguments

```
-javaagent:/opt/btm/WebSphere1_9080/eg_btm.jar -
DEG_PROPS_HOME=/opt/btm/WebSphere1_9080
```

Executable JAR file name

Disable JIT

Operating system name: windows

Figure 2.16: Configuring the JVM arguments for a WebSphere Server on Unix

25. Here, specify the following:

```
-javaagent:<<PATH TO THE eg_btm.jar FILE>>
```

```
-DEG_PROPS_HOME=<<PATH OF THE LOCAL FOLDER CONTAINING THE .PROPS FILES>>
```

For instance, if the eg_ btm.jar file and .props files have been copied to the **/opt/egurkha/lib/btm/Websphere1_9080** directory, the specification will be: :

```
-javaagent:/opt/btm/WebSphere1_9080/eg_btm.jar
```

```
-DEG_PROPS_HOME=/opt/btm/WebSphere1_9080
```

26. Moreover, in Unix environments, if the eG agent is deployed on the same host as the WebSphere server, then both the agent and the server will be running using different user privileges. In this situation, by default, the eG Java BTM logs will not be created. In order to create the same, insert the following entry after the -DEG_LOG_HOME specification .

```
-DEG_LOG_HOME=<LogFile_Path>
```

Before providing this specification, make sure you create a folder for BTM logs - say, eGBTMLogs - in any directory to which the target application server has access. Then, against, -

DEG_LOG_HOME, provide the full path to the eGBTMLogs directory. Where multiple instances on the same server are to be BTM-enabled, you can use the same directory for writing log files of all instances.

For example, to create log files in the **/App001/eGBTMLogs** directory, the complete specification will be as follows:

```
-javaagent:/opt/btm/WebSphere1_9080/eg_btm.jar
```

```
-DEG_PROPS_HOME=/opt/btm/WebSphere1_9080
```

```
-DEG_LOG_HOME=/App001/eGBTMLogs
```

27. Save the changes and restart the WebSphere server.
28. Where multiple instances of WebSphere are monitored, make sure you perform step 17- 27 above for each WebSphere server instance.

2.4 Installing eG Java BTM on an Oracle WebLogic Server

The steps for deploying the eG Java BTM on a WebLogic server will differ based on the platform on which the target WebLogic server is running - whether on a Windows platform or a Unix platform.

2.4.1 BTM-Enabling a WebLogic Server Running on a Windows Platform

If the WebLogic server is running on a Windows operating system, then follow the steps below to BTM-enable that server:

1. Login to the WebLogic server.
2. Open a browser on the server, connect to the eG manager, and login to the eG admin interface.
3. Manage the WebLogic server as a separate component using the eG administrative interface. When managing, make a note of the **Nick name** and **Port number** that you provide.
4. If multiple WebLogic server instances are operating on a single host, and you want to BTM-enable all the instances, then you will have to manage each instance as a separate component using the eG administrative interface. When doing so, make a note of the **Nick name** and **Port number** using which you managed each instance.
5. Next, log out of the eG admin interface. Then, create a **btm** directory anywhere on the WebLogic server - say, **C:\btm**. Under this directory, create a sub-folder. Make sure that you name this sub-folder in the following format: <*Managed_Component_NickName*>_<*Managed_Component_Port*>. For instance, if you have managed the WebLogic server using the nick name *weblogic1*

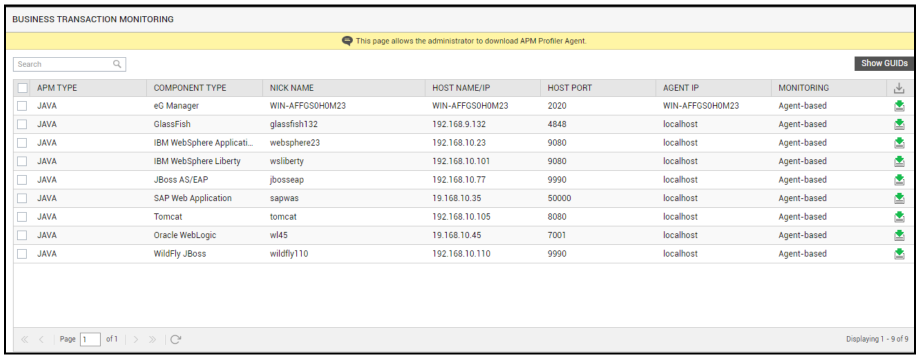
and the port number *7001*, the new directory under the **btm** directory should be named as *weblogic1_7001*.

6. If you have managed multiple WebLogic server instances running on a single host, then you will have to create multiple sub-directories under the **btm** directory- one each for every instance. Each of these sub-directories should be named after the **Nick name** and **Port number** using which the corresponding instance has been managed in eG.

7. Once the new sub-directory(ies) is created, open a browser on the WebLogic server, connect to the eG manager, and login to the eG admin interface again.

8. Follow the Agents -> BTM Profiler Download menu sequence in the eG admin interface.

9. Figure 2.17 will appear listing the servers that can be instrumented for APM by eG. In this list, locate the WebLogic server that you want to BTM-enable. Once you locate the server, click the **Download** icon corresponding to that server to download the **APM Profiler Agent** to that server. If multiple WebLogic server instances on a single host are managed, then you will have to download the APM Profiler Agent separately for each of the managed instances.


BUSINESS TRANSACTION MONITORING

This page allows the administrator to download APM Profiler Agent.

APM TYPE	COMPONENT TYPE	NICK NAME	HOST NAME/IP	HOST PORT	AGENT IP	MONITORING
JAVA	eG Manager	WIN-AFFGS0H0M23	WIN-AFFGS0H0M23	2020	WIN-AFFGS0H0M23	Agent-based
JAVA	GlassFish	glassfish132	192.168.9.132	4848	localhost	Agent-based
JAVA	IBM WebSphere Applicati...	websphere23	192.168.10.23	9080	localhost	Agent-based
JAVA	IBM WebSphere Liberty	wsliberty	192.168.10.101	9080	localhost	Agent-based
JAVA	JBoss AS/EAP	jbosseap	192.168.10.77	9990	localhost	Agent-based
JAVA	SAP Web Application	sapwas	19.168.10.35	50000	localhost	Agent-based
JAVA	Tomcat	tomcat	192.168.10.105	8080	localhost	Agent-based
JAVA	Oracle WebLogic	wl45	19.168.10.45	7001	localhost	Agent-based
JAVA	WildFly JBoss	wildfly110	192.168.10.110	9990	localhost	Agent-based

Page 1 of 1 — Displaying 1 - 9 of 9

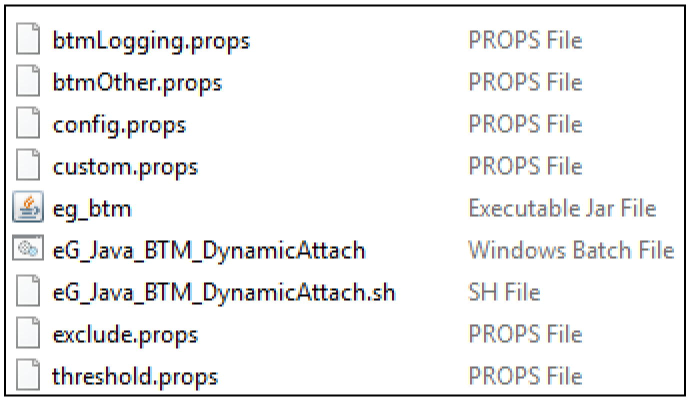
Figure 2.17: Downloading the APM Profiler Agent for the WebLogic server

10. Upon clicking the **Download** icon in Figure 2.17, a zip file named **javaagent_<Nick_name_of_ WebLogic_ server>_ <Port_ number_ of_ WebLogic_ server** will get downloaded. For instance, if you have managed the WebLogic server using the nickname 'weblogic1' and the port number '7001', then the name of the zip file will be *javaagent_weblogic1_7001*. Where multiple WebLogic server instances have been managed, you will be downloading multiple zip files - one each for every instance. The names of these zip files will automatically carry the nick name and port number you assigned to the corresponding server instance.

11. Copy the downloaded zip file(s) to the corresponding sub-directory(ies) of the **btm** directory (see steps 5 and 6 above). For example, the zip file named *javaagent_weblogic1_7001*, should be copied to the **C:\btm\weblogic1_7001** directory .

12. Extract the contents of each zip file into the same sub-directory to which that zip file was copied.

13. Figure 2.18 depicts the extracted contents of the zip file.



btmLogging.props	PROPS File
btmOther.props	PROPS File
config.props	PROPS File
custom.props	PROPS File
eg_btm	Executable Jar File
eG_Java_BTM_DynamicAttach	Windows Batch File
eG_Java_BTM_DynamicAttach.sh	SH File
exclude.props	PROPS File
threshold.props	PROPS File

Figure 2.18: Contents of the APM Profiler Agent zip

14. From Figure 2.18, it is evident that the zip file contains an **eg_btm.jar** file and a few property files, namely - **btmOther.props** , **btmLogging.props** , **config.props** , **custom.props** , **exclude.props**, and **threshold.props** files.

15. Next, proceed to BTM-enable the WebLogic server/instance. For that, edit the **btmOther.props** file in the sub-directory (of the **btm** directory) that corresponds to that WebLogic server/ instance. You will find the following lines in the file:

```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Below property is BTM Server Socket Port, through which eG Agent Communicates
# Restart is required, if any changes in this property
# Default port is "13931"
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#
BTM_Port=13931
#
```

By default, the **BTMPort** parameter is set to 13931. If you want to enable eG Java BTM on a different port, then specify the same here. In this case, when configuring the **Java Business Transactions** test or the **Key Java Business Transactions** test for that application server, make sure you configure the **BTM** port parameter of the test with this port number.

Note:

When BTM-enabling multiple instances on the same server, make sure you configure a different **BTM Port** for each instance.

Also, by default, the **Designated_Agent** parameter will be empty; do not disturb this default setting. In this case therefore, the eG Java BTM will treat the host from which the very first 'measure request' comes in as the **Designated_Agent**.

Note:

In case a specific **Designated_Agent** is not provided, and the eG Java BTM treats the host from which the very first 'measure request' comes in as the **Designated_Agent**, then if such a **Designated_Agent** is stopped or uninstalled for any reason, the eG Java BTM will wait for a maximum of 10 measure periods for that 'deemed' **Designated_Agent** to request for metrics. If no requests come in for 10 consecutive measure periods, then the eG Java BTM will begin responding to 'measure requests' coming in from any other eG agent.

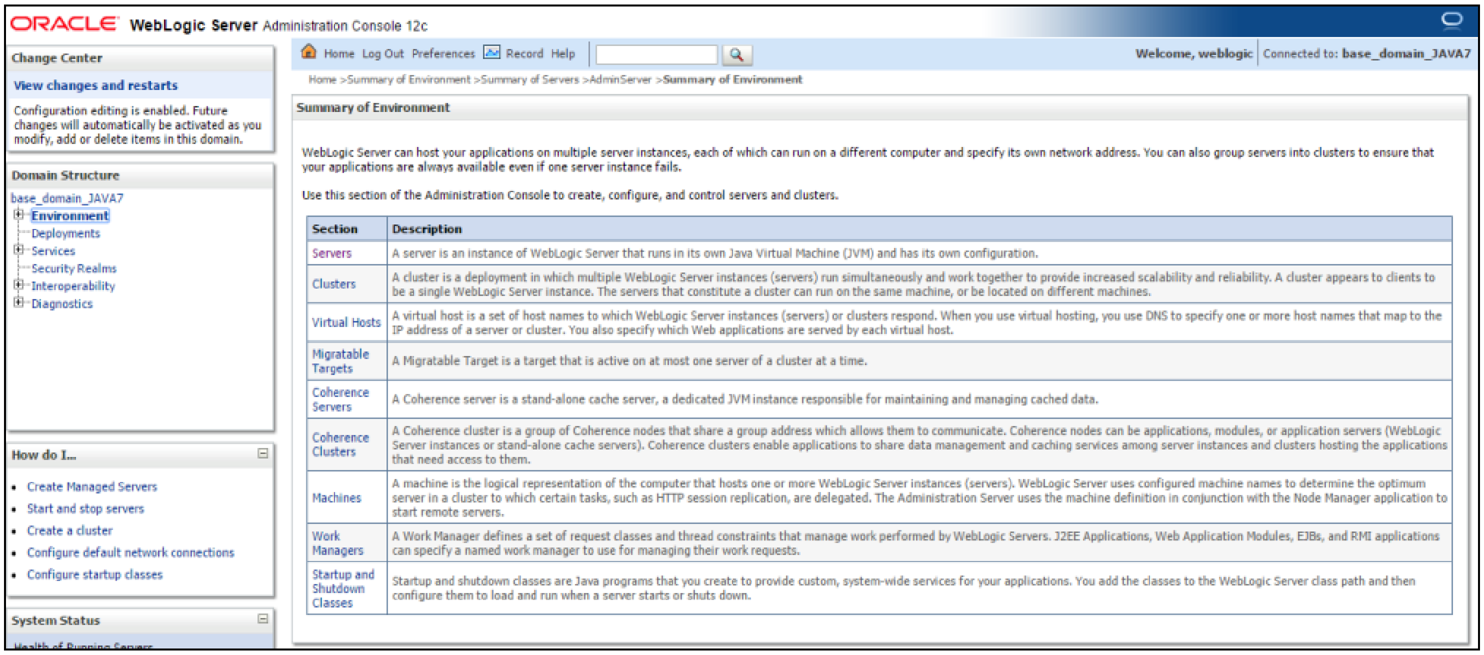
16. Finally, save the **btmOther.props** file.

17. Then, you need to configure the WebLogic server/server instance with the path to the **eg_ btm.jar** and **.props** files. The procedure to achieve this varies, depending upon the following:

    - Whether you want to BTM-enable a stand-alone WebLogic server/server instance
    - Whether you want to BTM-enable the Admin server of a WebLogic cluster

18. To BTM-enable a stand-alone WebLogic server/server instance, first login to the WebLogic Administration console. Then, follow the steps detailed below:

- When Figure 2.19 appears, click on the **Servers** link in the right panel.



ORACLE WebLogic Server Administration Console 12c

Home Log Out Preferences Record Help | Welcome, weblogic | Connected to: base_domain_JAVA7

Home >Summary of Environment >Summary of Servers >AdminServer >**Summary of Environment**

Change Center

View changes and restarts

Configuration editing is enabled. Future changes will automatically be activated as you modify, add or delete items in this domain.

Domain Structure

base_domain_JAVA7
- Environment
- Deployments
- Services
- Security Realms
- Interoperability
- Diagnostics

How do I...

- Create Managed Servers
- Start and stop servers
- Create a cluster
- Configure default network connections
- Configure startup classes

System Status

Health of Running Servers

Summary of Environment

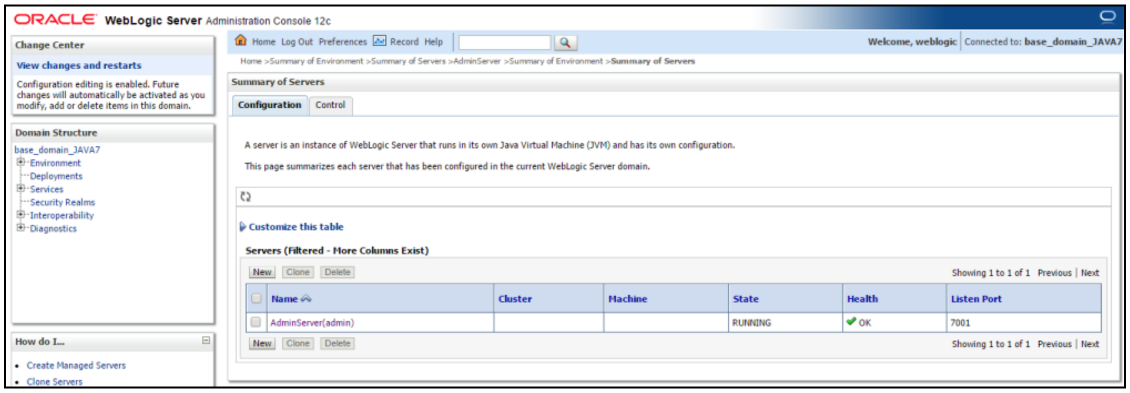
WebLogic Server can host your applications on multiple server instances, each of which can run on a different computer and specify its own network address. You can also group servers into clusters to ensure that your applications are always available even if one server instance fails.

Use this section of the Administration Console to create, configure, and control servers and clusters.

Section	Description
Servers	A server is an instance of WebLogic Server that runs in its own Java Virtual Machine (JVM) and has its own configuration.
Clusters	A cluster is a deployment in which multiple WebLogic Server instances (servers) run simultaneously and work together to provide increased scalability and reliability. A cluster appears to clients to be a single WebLogic Server instance. The servers that constitute a cluster can run on the same machine, or be located on different machines.
Virtual Hosts	A virtual host is a set of host names to which WebLogic Server instances (servers) or clusters respond. When you use virtual hosting, you use DNS to specify one or more host names that map to the IP address of a server or cluster. You also specify which Web applications are served by each virtual host.
Migratable Targets	A Migratable Target is a target that is active on at most one server of a cluster at a time.
Coherence Servers	A Coherence server is a stand-alone cache server, a dedicated JVM instance responsible for maintaining and managing cached data.
Coherence Clusters	A Coherence cluster is a group of Coherence nodes that share a group address which allows them to communicate. Coherence nodes can be applications, modules, or application servers (WebLogic Server instances or stand-alone cache servers). Coherence clusters enable applications to share data management and caching services among server instances and clusters hosting the applications that need access to them.
Machines	A machine is the logical representation of the computer that hosts one or more WebLogic Server instances (servers). WebLogic Server uses configured machine names to determine the optimum server in a cluster to which certain tasks, such as HTTP session replication, are delegated. The Administration Server uses the machine definition in conjunction with the Node Manager application to start remote servers.
Work Managers	A Work Manager defines a set of request classes and thread constraints that manage work performed by WebLogic Servers. J2EE Applications, Web Application Modules, EJBs, and RMI applications can specify a named work manager to use for managing their work requests.
Startup and Shutdown Classes	Startup and shutdown classes are Java programs that you create to provide custom, system-wide services for your applications. You add the classes to the WebLogic Server class path and then configure them to load and run when a server starts or shuts down.

Figure 2.19: Clicking on the Servers link

- Figure 2.20 will then appear. Here, click on the server instance to be BTM-enabled.



ORACLE WebLogic Server Administration Console 12c

Home Log Out Preferences Record Help | Welcome, weblogic | Connected to: base_domain_JAVA7

Home >Summary of Environment >Summary of Servers >AdminServer >Summary of Environment >**Summary of Servers**

Change Center

View changes and restarts

Configuration editing is enabled. Future changes will automatically be activated as you modify, add or delete items in this domain.

Domain Structure

base_domain_JAVA7
- Environment
- Deployments
- Services
- Security Realms
- Interoperability
- Diagnostics

How do I...

- Create Managed Servers
- Clone Servers

Summary of Servers

Configuration | Control

A server is an instance of WebLogic Server that runs in its own Java Virtual Machine (JVM) and has its own configuration.

This page summarizes each server that has been configured in the current WebLogic Server domain.

Customize this table

Servers (Filtered - More Columns Exist)

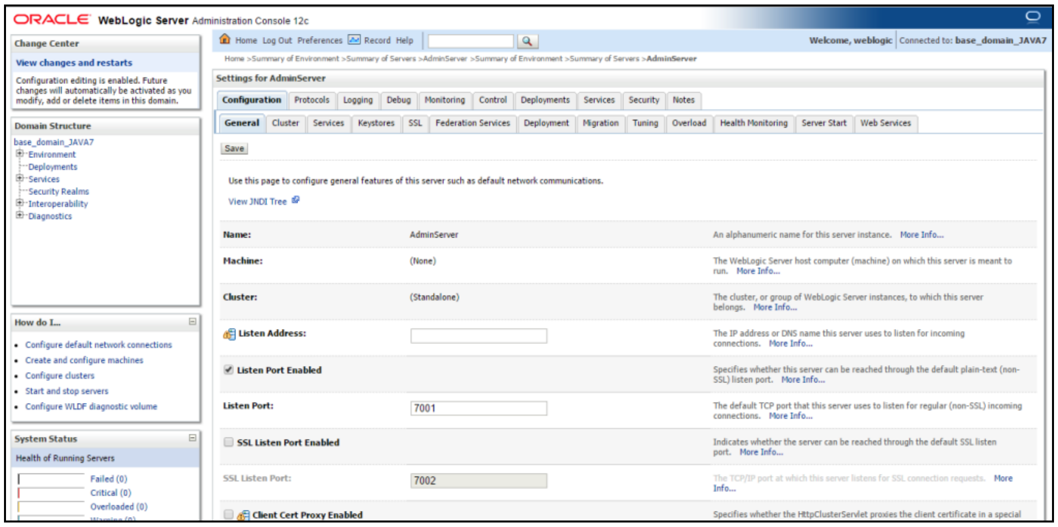
New | Clone | Delete — Showing 1 to 1 of 1 Previous | Next

	Name	Cluster	Machine	State	Health	Listen Port
☐	AdminServer(admin)			RUNNING	✓ OK	7001

New | Clone | Delete — Showing 1 to 1 of 1 Previous | Next

Figure 2.20: Clicking on the server instance to be BTM-enabled

- Figure 2.21 will then appear.



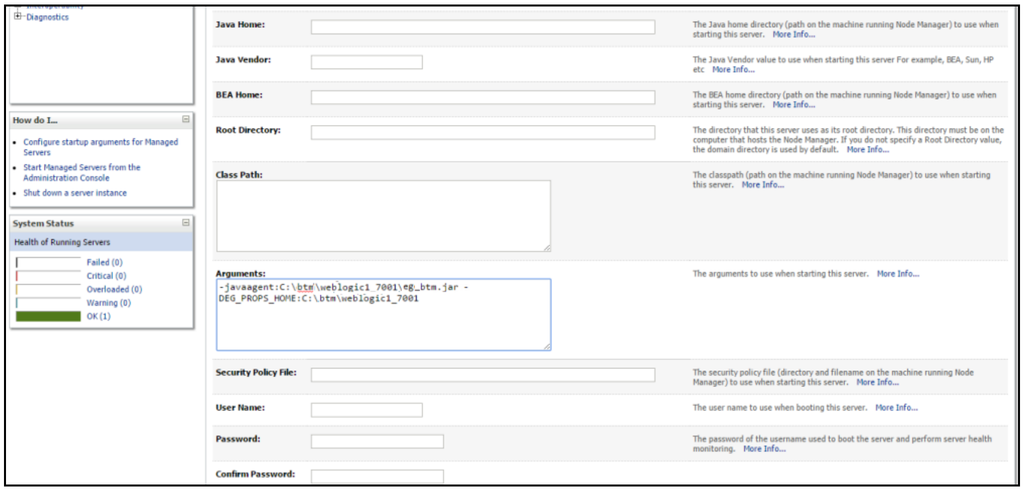
Configuration editing is enabled. Future changes will automatically be activated as you modify, add or delete items in this domain.

Use this page to configure general features of this server such as default network communications.

Name:	AdminServer	An alphanumeric name for this server instance. More Info...
Machine:	(None)	The WebLogic Server host computer (machine) on which this server is meant to run. More Info...
Cluster:	(Standalone)	The cluster, or group of WebLogic Server instances, to which this server belongs. More Info...
Listen Address:		The IP address or DNS name this server uses to listen for incoming connections. More Info...
✓ Listen Port Enabled		Specifies whether this server can be reached through the default plain-text (non-SSL) listen port. More Info...
Listen Port:	7001	The default TCP port that this server uses to listen for regular (non-SSL) incoming connections. More Info...
SSL Listen Port Enabled		Indicates whether the server can be reached through the default SSL listen port. More Info...
SSL Listen Port:	7002	The TCP/IP port at which this server listens for SSL connection requests. More Info...
Client Cert Proxy Enabled		Specifies whether the HttpClusterServlet proxies the client certificate in a special

Figure 2.21: Viewing the configuration of the chosen server instance

- Keep scrolling down the right panel of Figure 2.21 until the **Arguments** text box comes into view (see Figure 2.22).


Diagnostics
How do I...
Configure startup arguments for Managed Servers
Start Managed Servers from the Administration Console
Shut down a server instance
System Status
Health of Running Servers
Failed (0)
Critical (0)
Overloaded (0)
Warning (0)
OK (1)

Java Home:		The Java home directory (path on the machine running Node Manager) to use when starting this server. More Info...
Java Vendor:		The Java Vendor value to use when starting this server For example, BEA, Sun, HP etc More Info...
BEA Home:		The BEA home directory (path on the machine running Node Manager) to use when starting this server. More Info...
Root Directory:		The directory that this server uses as its root directory. This directory must be on the computer that hosts the Node Manager. If you do not specify a Root Directory value, the domain directory is used by default. More Info...
Class Path:		The classpath (path on the machine running Node Manager) to use when starting this server. More Info...
Arguments:	-javaagent:C:\btm\weblogic1_7001\eg_btm.jar -DEG_PROPS_HOME:C:\btm\weblogic1_7001	The arguments to use when starting this server. More Info...
Security Policy File:		The security policy file (directory and filename on the machine running Node Manager) to use when starting this server. More Info...
User Name:		The user name to use when booting this server. More Info...
Password:		The password of the username used to boot the server and perform server health monitoring. More Info...
Confirm Password:		

Figure 2.22: Configuring the JVM arguments

- In the **Arguments** text box, specify the following lines:

```
-javaagent:<Path_to_eg_btm.jar_file>
```

```
-DEG_PROPS_HOME=<Path_to_the_folder_containing_the_.props_files>
```

For instance, if the eg_btm.jar file and .props files are extracted into the **C:\btm\weblogic1_7001** directory, the above specification will be:

```
-javaagent:C:\btm\weblogic_7001\eg_btm.jar
```

```
-DEG_PROPS_HOME=C:\btm\weblogic_7001
```

- Finally, save the changes and restart the WebLogic server.

19. Where multiple instances of WebLogic are monitored, make sure you perform step 15-18 above for each WebLogic server instance.

20. To BTM-enable the Admin server of a WebLogic cluster, you need to edit the start-up script of the Admin server. For that, follow the steps below:

    - Login to the Admin server, open the start-up script, and insert the following lines in it:

```
set EG_JAVA_OPTIONS_ADMIN_SERVER="-javaagent:<Path_to_eg_btm.jar_file> -DEG_PROPS_
HOME=<Path_to_the_folder_containin_.props_files>"
```

```
if "%SERVER_NAME%"=="AdminServer" (
```

```
set EG_JAVA_OPTIONS=%EG_JAVA_OPTIONS_ADMIN_SERVER%
```

```
)
```

```
set JAVA_OPTIONS=%JAVA_OPTIONS% %JAVA_PROPERTIES% -
Dwlw.iterativeDev=%iterativeDevFlag% -Dwlw.testConsole=%testConsoleFlag% -
Dwlw.logErrorsToConsole=%logErrorsToConsoleFlag% %EG_JAVA_OPTIONS%
```

For instance, if the eg_btm.jar file and .props files are extracted into the **C:\btm\WebLogic_7001** directory, the above specification will be:

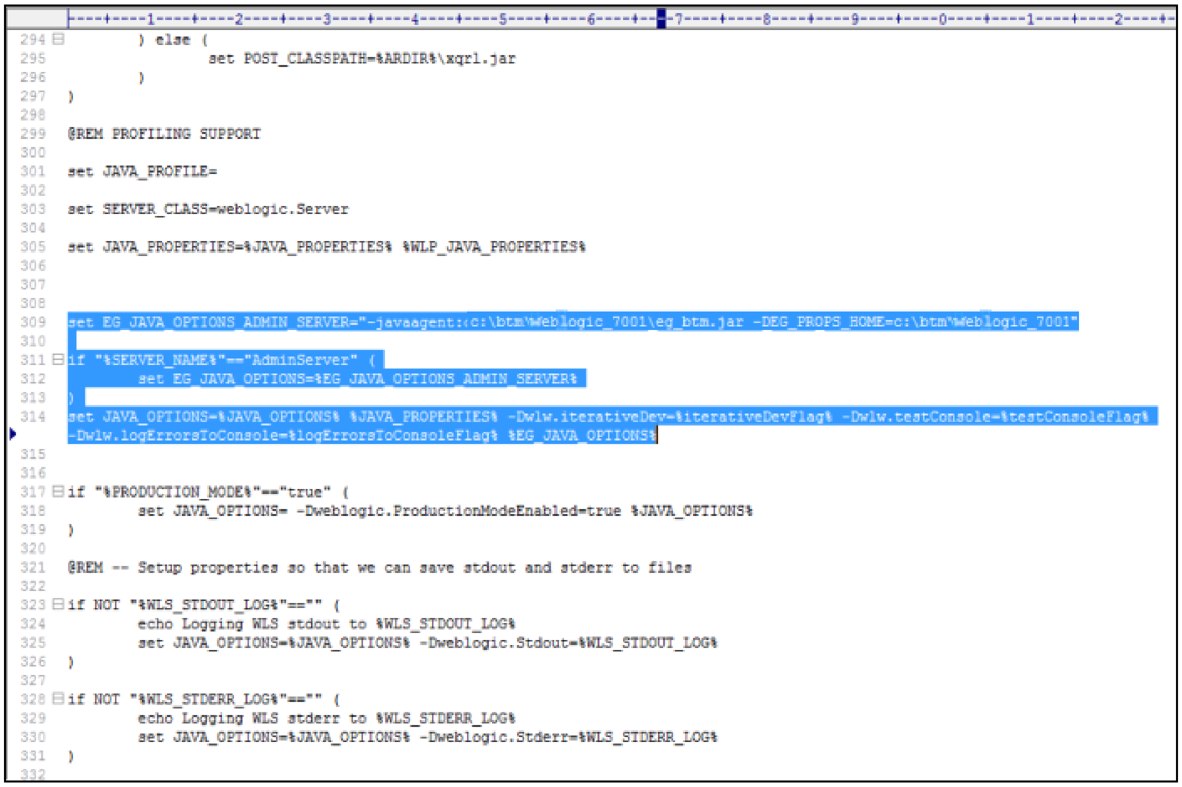
```
set EG_JAVA_OPTIONS_ADMIN_SERVER="-javaagent:c:\btm\WebLogic_7001\eg_btm.jar -DEG_
PROPS_HOME=c:\btm\WebLogic_7001"
```

```
if "%SERVER_NAME%"=="AdminServer" (
```

```
set EG_JAVA_OPTIONS=%EG_JAVA_OPTIONS_ADMIN_SERVER%
```

```
)
```

```
set JAVA_OPTIONS=%JAVA_OPTIONS% %JAVA_PROPERTIES% -
Dwlw.iterativeDev=%iterativeDevFlag% -Dwlw.testConsole=%testConsoleFlag% -
Dwlw.logErrorsToConsole=%logErrorsToConsoleFlag% %EG_JAVA_OPTIONS%
```



```
        ) else (
                set POST_CLASSPATH=%ARDIR%\xqrl.jar
        )
)

@REM PROFILING SUPPORT

set JAVA_PROFILE=

set SERVER_CLASS=weblogic.Server

set JAVA_PROPERTIES=%JAVA_PROPERTIES% %WLP_JAVA_PROPERTIES%



set EG_JAVA_OPTIONS_ADMIN_SERVER="-javaagent:c:\btm\weblogic_7001\eg_btm.jar -DEG_PROPS_HOME=c:\btm\weblogic_7001"

if "%SERVER_NAME%"=="AdminServer" (
        set EG_JAVA_OPTIONS=%EG_JAVA_OPTIONS_ADMIN_SERVER%
)
set JAVA_OPTIONS=%JAVA_OPTIONS% %JAVA_PROPERTIES% -Dwlw.iterativeDev=%iterativeDevFlag% -Dwlw.testConsole=%testConsoleFlag%
-Dwlw.logErrorsToConsole=%logErrorsToConsoleFlag% %EG_JAVA_OPTIONS%


if "%PRODUCTION_MODE%"=="true" (
        set JAVA_OPTIONS= -Dweblogic.ProductionModeEnabled=true %JAVA_OPTIONS%
)

@REM -- Setup properties so that we can save stdout and stderr to files

if NOT "%WLS_STDOUT_LOG%"=="" (
        echo Logging WLS stdout to %WLS_STDOUT_LOG%
        set JAVA_OPTIONS=%JAVA_OPTIONS% -Dweblogic.Stdout=%WLS_STDOUT_LOG%
)

if NOT "%WLS_STDERR_LOG%"=="" (
        echo Logging WLS stderr to %WLS_STDERR_LOG%
        set JAVA_OPTIONS=%JAVA_OPTIONS% -Dweblogic.Stderr=%WLS_STDERR_LOG%
)

```

Figure 2.23: Editing the start-up script of the WebLogic Admin server on Windows that is monitored in an agent-based manner

- Finally, save the file and restart the Admin server.

2.4.2 BTM-Enabling a WebLogic Server Running on a Unix Platform

If the target WebLogic server is running on a Unix operating system, then follow the steps below to BTM-enable that server:

1. Login to any system in your environment that supports a browser and has network access to the eG manager.
2. Open a browser on that system, connect to the eG manager, and login to the eG admin interface.
3. Manage the target WebLogic server as a separate component using the eG administrative interface. When managing, make a note of the **Nick name** and **Port number** that you provide.
4. If multiple WebLogic server instances are operating on a single host, and you want to monitor each of those instances, then you will have to manage each instance as a separate WebLogic server component using the eG administrative interface. When doing so, make a note of the **Nick name** and **Port number** using which you managed each instance.

5. Next, log out of the eG admin interface and the system.
6. Log into the target WebLogic server. Then, create a **btm** directory anywhere on the target server - say, **/opt/btm**. Under this directory, create a sub-folder. Make sure that you name this sub-folder in the following format: <*Managed_Component_NickName*>_<*Managed_Component_Port*>. For instance, if you have managed the WebLogic server using the nick name *weblogic1* and the port number *7001*, the new directory under the **btm** directory should be named as *tomcat1_8080*.
7. If you have managed multiple WebLogic server instances running on a single host, then you will have to create multiple sub-directories under the **btm** directory - one each for every instance. Each of these sub-directories should be named after the **Nick name** and **Port number** using which the corresponding instance has been managed in eG.
8. Once the new sub-directory(ies) is created, log out of the WebLogic server. Log back into the system you used in step 1 above. Open a browser on the system, connect to the eG manager, and login to the eG admin interface again.
9. Follow the Agents -> BTM Profiler Download menu sequence in the eG admin interface.
10. Figure 2.17 will appear listing the servers that can be instrumented for APM by eG. In this list, locate the WebLogic server that you want to BTM-enable. Once you locate the server, click the **Download** icon corresponding to that server to download the **APM Profiler Agent** to that server. If multiple WebLogic server instances on a single host are managed, then you will have to download the APM Profiler Agent separately for each of the managed instances.
11. Upon clicking the **Download** icon in Figure 2.17, a zip file named **javaagent_<Nick_name_of_WebLogic_ server>_ <Port_ number_ of_ WebLogic_ server** will get downloaded. For instance, if you have managed the WebLogic server using the nickname 'weblogic1' and the port number '7001', then the name of the zip file will be *javaagent_weblogic1_7001*. Where multiple WebLogic server instances have been managed, you will be downloading multiple zip files - one each for every instance. The names of these zip files will automatically carry the nick name and port number you assigned to the corresponding WebLogic server instance.
12. Next, using FTP tools like putty or WinSCP, transfer the downloaded zip file(s) to the corresponding sub-directory(ies) (of the **btm** directory), which you created on the WebLogic server in steps 6 and 7 above. For example, the zip file named *javaagent_weblogic1_7001*, should be transferred to the**/opt/btm/weblogic_7001** directory on the target WebLogic server.
13. Log out of the system and log back into the WebLogic server.
14. Extract the contents of each zip file into the same sub-directory to which that zip file was copied.

15. Figure 2.18 depicts the extracted contents of the zip file.

16. From Figure 2.18, it is evident that the zip file contains an **eg_btm.jar** file and a few property files, namely - **btmOther.props** , **btmLogging.props** , **config.props** , **custom.props** , **exclude.props**, and **threshold.props** files.

17. Next, proceed to BTM-enable the WebLogic server/instance. For that, edit the **btmOther.props** file in the sub-directory (of the **btm** directory) that corresponds to that server/ instance. You will find the following lines in the file:

```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Below property is BTM Server Socket Port, through which eG Agent Communicates
# Restart is required, if any changes in this property
# Default port is "13931"
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#
BTM_Port=13931
#
```

By default, the **BTMPort** parameter is set to 13931. If you want to enable eG Java BTM on a different port, then specify the same here. In this case, when configuring the **Java Business Transactions** test or the **Key Java Business Transactions** test for that application server, make sure you configure the **BTM PORT** parameter of the test with this port number.

Note:

When BTM-enabling multiple instances on the same server, make sure you configure a different **BTM Port** for each instance.

Also, by default, the **Designated_Agent** parameter will be empty; do not disturb this default setting. In this case therefore, the eG Java BTM will treat the host from which the very first 'measure request' comes in as the **Designated_Agent**.

Note:

In case a specific **Designated_Agent** is not provided, and the eG Java BTM treats the host from which the very first 'measure request' comes in as the **Designated_Agent**, then if such a **Designated_Agent** is stopped or uninstalled for any reason, the eG Java BTM will wait for a maximum of 10 measure periods for that 'deemed' **Designated_Agent** to request for metrics. If no requests come in for 10 consecutive measure periods, then the eG Java BTM will begin responding to 'measure requests' coming in from any other eG agent.

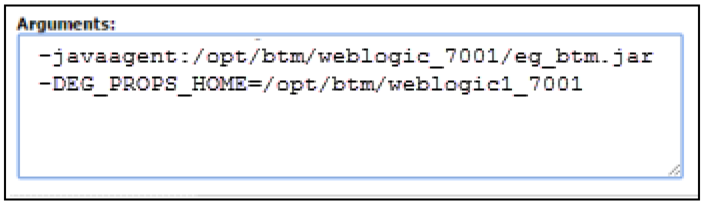
18. Finally, save the **btmOther.props** file.

19. Then, you need to configure the WebLogic server/server instance with the path to the **eg_ btm.jar** and **.props** files. The procedure to achieve this varies, depending upon the following:

    - Whether you want to BTM-enable a stand-alone WebLogic server/server instance
    - Whether you want to BTM-enable the Admin server of a WebLogic cluster

20. To BTM-enable a stand-alone WebLogic server/server instance, first login to the WebLogic Administration console. Then, follow the steps detailed below:

    - When Figure 2.19 appears, click on the **Servers** link in the right panel.
    - Figure 2.20 will then appear. Here, click on the server instance to be BTM-enabled.
    - Figure 2.21 will then appear.
    - Keep scrolling down the right panel of Figure 2.21 until the **Arguments** text box comes into view (see Figure 2.22).



```
-javaagent:/opt/btm/weblogic_7001/eg_btm.jar
-DEG_PROPS_HOME=/opt/btm/weblogic1_7001
```

Figure 2.24: Configuring the JVM arguments for a WebLogic server on Unix

- In the **Arguments** text box, specify the following lines:

```
-javaagent:<Path_to_eg_btm.jar_file>
```

```
-DEG_PROPS_HOME=<Path_to_the_folder_containing_the_.props_files>
```

For instance, if the eg_btm.jar file and .props files are extracted into the **/opt/btm/weblogic1_ 7001** directory, the specification will be:

```
-javaagent:/opt/btm/weblogic1_7001/eg_btm.jar
```

```
-DEG_PROPS_HOME=/opt/btm/weblogic1_7001
```

- Additionally, in Unix environments, if the eG agent is deployed on the same host as the

WebLogic server, then both the agent and the server will be running using different user privileges. In this situation, by default, the eG Java BTM logs will not be created. In order to create the same, insert the following entry after the -DEG_PROPS_HOME specification .

```
-DEG_LOG_HOME=<LogFile_Path>
```

Before providing this specification, make sure you create a folder for BTM logs - say, **eGBTMLogs** - in any directory to which the target application server has access. Then, against, -DEG_LOG_HOME, provide the full path to the **eGBTMLogs** directory. Where multiple instances on the same server are to be BTM-enabled, you can use the same directory for writing log files of all instances.

For example, to create log files in the **/App001/eGBTMLogs** directory, the complete specification will be as follows:

```
-javaagent:/opt/btm/weblogic1_7001/eg_btm.jar
```

```
-DEG_PROPS_HOME=/opt/btm/webLogic_7001
```

```
-DEG_LOG_HOME=/App001/eGBTMLogs
```

- Finally, save the changes and restart the WebLogic server.

21. Where multiple instances of WebLogic are monitored, make sure you perform steps 17-20 above for each WebLogic server instance.
22. To BTM-enable the Admin server of a WebLogic cluster, you need to edit the start-up script of the Admin server. For that, follow the steps below:

- Login to the Admin server, open the start-up script, and insert the following lines in it:

```
set EG_JAVA_OPTIONS_ADMIN_SERVER="-javaagent:<Path_to_eg_btm.jar_file> -DEG_PROPS_
HOME=<Path_to_the_folder_containin_.props_files>"
```

```
if "%SERVER_NAME%"=="AdminServer" (
```

```
set EG_JAVA_OPTIONS=%EG_JAVA_OPTIONS_ADMIN_SERVER%
```

```
)
```

```
set JAVA_OPTIONS=%JAVA_OPTIONS% %JAVA_PROPERTIES% -
Dwlw.iterativeDev=%iterativeDevFlag% -Dwlw.testConsole=%testConsoleFlag% -
Dwlw.logErrorsToConsole=%logErrorsToConsoleFlag% %EG_JAVA_OPTIONS%
```

For instance, if the eg_btm.jar file and .props files are extracted into the **/opt/btm/webLogic1_7001** directory, then your specification will be as follows:

```
EG_JAVA_OPTIONS_ADMIN_SERVER="-javaagent:/opt/btm/webLogic_7001/eg_btm.jar -DEG_
PROPS_HOME=/opt/btm/webLogic_7001"
```

- If the eG agent and the Admin server are co-hosted on the same Unix host, then to create the log files, insert the following entry after the -DEG_PROPS_HOME specification and before the closing quotes.

```
-DEG_LOG_HOME=<LogFile_Path>
```

  Before providing this specification, make sure you create a folder for BTM logs - say, **eGBTMLogs** - in any directory to which the target application server has access. Then, against, -DEG_LOG_HOME, provide the full path to the **eGBTMLogs** directory. Where multiple instances on the same server are to be BTM-enabled, you can use the same directory for writing log files of all instances.

  For example, to create log files in the **/App001/eGBTMLogs** directory, the complete specification will be as follows:

```
EG_JAVA_OPTIONS_ADMIN_SERVER="-javaagent:/opt/btm/weblogic1_7001/eg_btm.jar -DEG_
PROPS_HOME=/opt/btm/webLogic_7001 -DEG_LOG_HOME=/App001/eGBTMLogs"
if [ "${SERVER_NAME}" = "AdminServer" ] ; then
EG_JAVA_OPTIONS="${EG_JAVA_OPTIONS_ADMIN_SERVER}"
fi
SAVE_JAVA_OPTIONS="${JAVA_OPTIONS} ${EG_JAVA_OPTIONS}"
```

- Finally, save the file and restart the Admin server.

2.5 Installing eG Java BTM on GlassFish

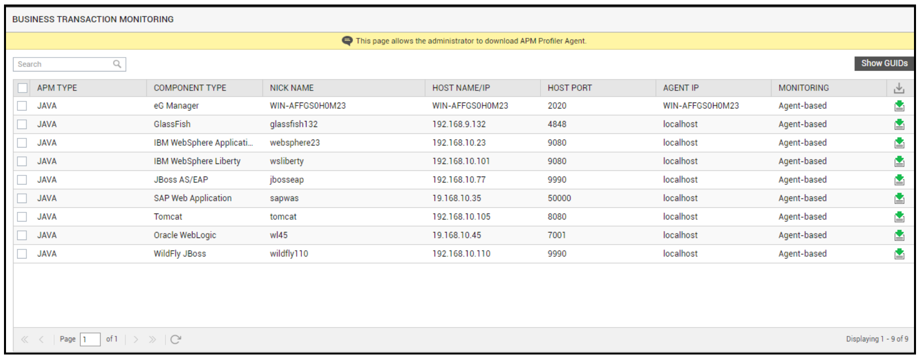
The steps for deploying the eG Java BTM on a GlassFish server will differ based on the platform on which the target GlassFish server is running - whether on a Windows platform or a Unix platform.

2.5.1 BTM-Enabling a GlassFish Server Running on a Windows Platform

If the GlassFish server is running on a Windows operating system, then follow the steps below to BTM-enable that server:

1. Login to the GlassFish server.
2. Open a browser on the server, connect to the eG manager, and login to the eG admin interface.

3. Manage the GlassFish server as a separate component using the eG administrative interface. When managing, make a note of the **Nick name** and **Port number** that you provide.
4. If multiple GlassFish server instances are operating on a single host, and you want to BTM-enable all the instances, then you will have to manage each instance as a separate component using the eG administrative interface. When doing so, make a note of the **Nick name** and **Port number** using which you managed each instance.
5. Next, log out of the eG admin interface. Then, create a **btm** directory anywhere on the GlassFish server - say, **C:\btm**. Under this directory, create a sub-folder. Make sure that you name this sub-folder in the following format: <*Managed_Component_NickName*>_<*Managed_Component_Port*>. For instance, if you have managed the GlassFish server using the nick name *WebSphere1* and the port number *9080*, the new directory under the **btm** directory should be named as *WebSphere1_9080*.
6. If you have managed multiple GlassFish server instances running on a single host, then you will have to create multiple sub-directories under the **btm** directory- one each for every instance. Each of these sub-directories should be named after the **Nick name** and **Port number** using which the corresponding instance has been managed in eG.
7. Once the new sub-directory(ies) is created, open a browser on the GlassFish server, connect to the eG manager, and login to the eG admin interface again.
8. Follow the Agents -> BTM Profiler Download menu sequence in the eG admin interface.
9. Figure 2.25 will appear listing the servers that can be instrumented for APM by eG. In this list, locate the GlassFish server that you want to BTM-enable. Once you locate the server, click the **Download** icon corresponding to that server to download the **APM Profiler Agent** to that server. If multiple GlassFish server instances on a single host are managed, then you will have to download the APM Profiler Agent separately for each of the managed instances.



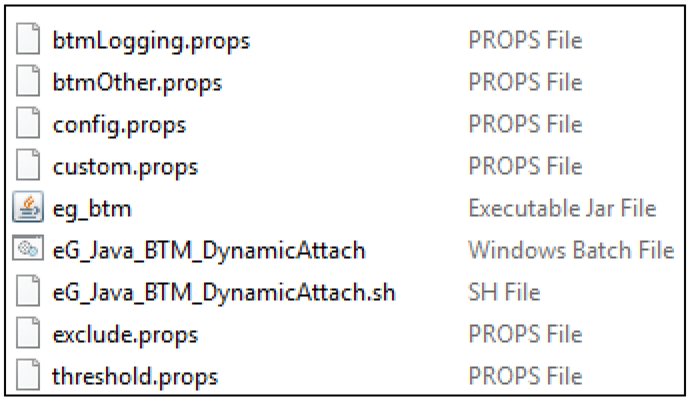
BUSINESS TRANSACTION MONITORING

This page allows the administrator to download APM Profiler Agent.

APM TYPE	COMPONENT TYPE	NICK NAME	HOST NAME/IP	HOST PORT	AGENT IP	MONITORING	
JAVA	eG Manager	WIN-AFFGS0H0M23	WIN-AFFGS0H0M23	2020	WIN-AFFGS0H0M23	Agent-based	
JAVA	GlassFish	glassfish132	192.168.9.132	4848	localhost	Agent-based	
JAVA	IBM WebSphere Applicati...	websphere23	192.168.10.23	9080	localhost	Agent-based	
JAVA	IBM WebSphere Liberty	wsliberty	192.168.10.101	9080	localhost	Agent-based	
JAVA	JBoss AS/EAP	jbosseap	192.168.10.77	9990	localhost	Agent-based	
JAVA	SAP Web Application	sapwas	19.168.10.35	50000	localhost	Agent-based	
JAVA	Tomcat	tomcat	192.168.10.105	8080	localhost	Agent-based	
JAVA	Oracle WebLogic	wl45	19.168.10.45	7001	localhost	Agent-based	
JAVA	WildFly JBoss	wildfly110	192.168.10.110	9990	localhost	Agent-based	

Figure 2.25: Downloading the APM Profiler Agent for the GlassFish server

10. Upon clicking the **Download** icon in Figure 2.25, a zip file named **javaagent_<Nick_name_of_ GlassFish_ server>_ <Port_ number_ of_ GlassFish_ server** will get downloaded. For instance, if you have managed the GlassFish server using the nickname 'GlassFish1' and the port number '4848', then the name of the zip file will be *javaagent_GlassFish1_4848*. Where multiple GlassFish server instances have been managed, you will be downloading multiple zip files - one each for every instance. The names of these zip files will automatically carry the nick name and port number you assigned to the corresponding server instance.
11. Copy the downloaded zip file(s) to the corresponding sub-directory(ies) of the **btm** directory (see steps 5 and 6 above). For example, the zip file named *javaagent_GlassFish1_4848*, should be copied to the **C:\btm\GlassFish1_4848** directory .
12. Extract the contents of each zip file into the same sub-directory to which that zip file was copied.
13. Figure 2.26 depicts the extracted contents of the zip file.



btmLogging.props	PROPS File
btmOther.props	PROPS File
config.props	PROPS File
custom.props	PROPS File
eg_btm	Executable Jar File
eG_Java_BTM_DynamicAttach	Windows Batch File
eG_Java_BTM_DynamicAttach.sh	SH File
exclude.props	PROPS File
threshold.props	PROPS File

Figure 2.26: Contents of the APM Profiler Agent zip

14. From Figure 2.26, it is evident that the zip file contains an **eg_btm.jar** file and a few property files, namely - **btmOther.props** , **btmLogging.props** , **config.props** , **custom.props** , **exclude.props**, and **threshold.props** files.
15. Next, proceed to BTM-enable the GlassFish server/instance. For that, edit the **btmOther.props** file in the sub-directory (of the **btm** directory) that corresponds to that GlassFish server/ instance. You will find the following lines in the file:

```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Below property is BTM Server Socket Port, through which eG Agent Communicates
# Restart is required, if any changes in this property
# Default port is "13931"
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
#
BTM_Port=13931
#
```

By default, the **BTMPort** parameter is set to 13931. If you want to enable eG Java BTM on a different port, then specify the same here. In this case, when configuring the **Java Business Transactions** test or the **Key Java Business Transactions** test for that application server, make sure you configure the **BTM** port parameter of the test with this port number.

Note:

When BTM-enabling multiple instances on the same server, make sure you configure a different **BTM Port** for each instance.

Also, by default, the **Designated_Agent** parameter will be empty; do not disturb this default setting. In this case therefore, the eG Java BTM will treat the host from which the very first 'measure request' comes in as the **Designated_Agent**.

Note:

In case a specific **Designated_Agent** is not provided, and the eG Java BTM treats the host from which the very first 'measure request' comes in as the **Designated_Agent**, then if such a **Designated_Agent** is stopped or uninstalled for any reason, the eG Java BTM will wait for a maximum of 10 measure periods for that 'deemed' **Designated_Agent** to request for metrics. If no requests come in for 10 consecutive measure periods, then the eG Java BTM will begin responding to 'measure requests' coming in from any other eG agent.

16. Finally, save the **btmOther.props** file.

17. Then, you need to configure the GlassFish server with the path to the **eg_btm.jar** and **.props** files. To achieve this, you can use one of the following two ways:

    - Through the GlassFish Administration console
    - By editing the start-up script of the GlassFish server instance

18. If you choose to use the GlassFish Administration console, then first, login to the console. Then, follow the steps detailed below:

    - When Figure 2.27 appears, click on the **server-config** node in the tree-structure in the left panel.

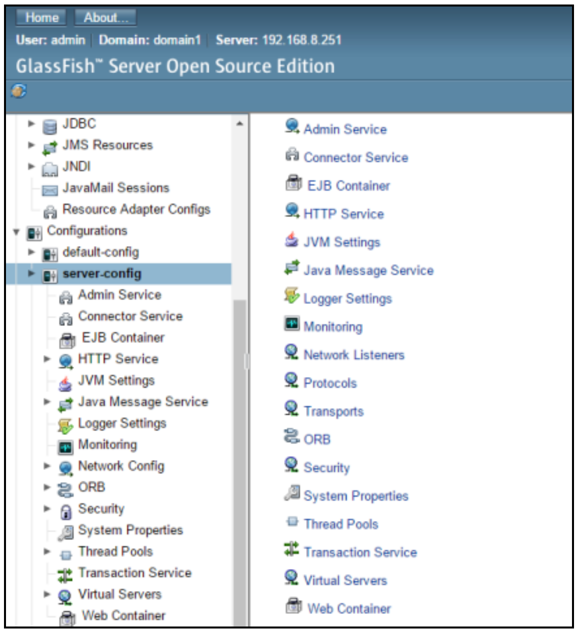
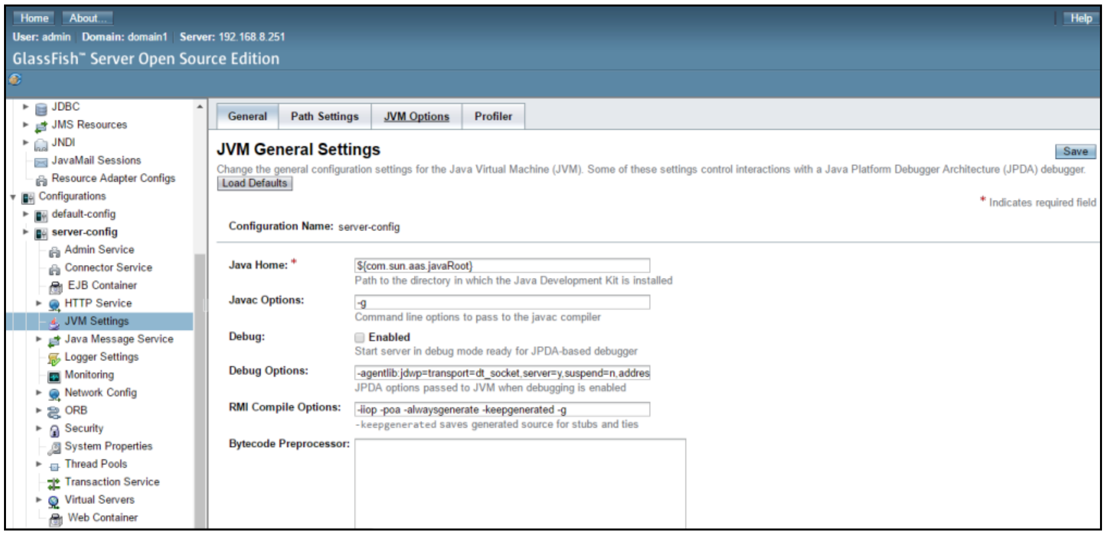


Figure 2.27: Clicking on the server-config node

- From the options listed in the right panel of Figure 2.27, select the **JVM Settings** option. Figure 2.28 will then appear. Select the **JVM Options** tab page in Figure 2.28.



JVM General Settings

Change the general configuration settings for the Java Virtual Machine (JVM). Some of these settings control interactions with a Java Platform Debugger Architecture (JPDA) debugger.

* Indicates required field

Configuration Name: server-config

Java Home: * ${com.sun.aas.javaRoot}
Path to the directory in which the Java Development Kit is installed

Javac Options: -g
Command line options to pass to the javac compiler

Debug: ☐ Enabled
Start server in debug mode ready for JPDA-based debugger

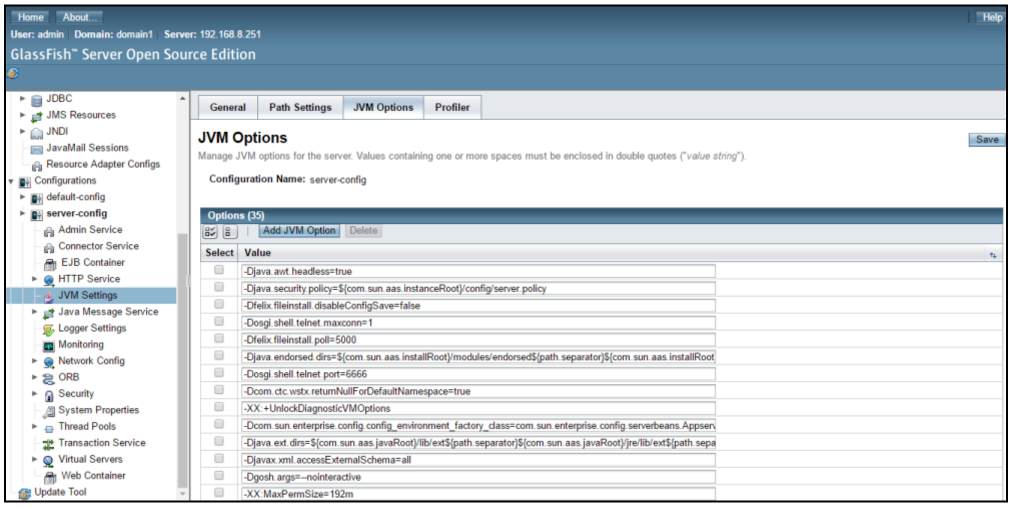
Debug Options: -agentlib:jdwp=transport=dt_socket,server=y,suspend=n,addres
JPDA options passed to JVM when debugging is enabled

RMI Compile Options: -iiop -poa -alwaysgenerate -keepgenerated -g
`-keepgenerated` saves generated source for stubs and ties

Bytecode Preprocessor:

Figure 2.28: Clicking on the JVM Options tab page

- Figure 2.29 will then appear. You now need to add two new JVM options. For this, click on the **Add JVM Option** button in Figure 2.29, twice.



JVM Options

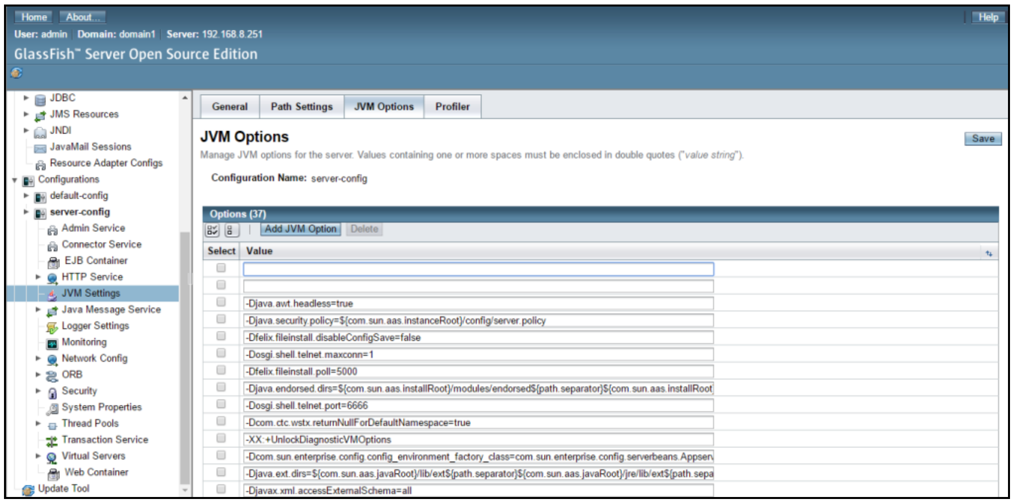
Manage JVM options for the server. Values containing one or more spaces must be enclosed in double quotes ("*value string*").

Configuration Name: server-config

```
-Djava.awt.headless=true
-Djava.security.policy=${com.sun.aas.instanceRoot}/config/server.policy
-Dfelix.fileinstall.disableConfigSave=false
-Dosgi.shell.telnet.maxconn=1
-Dfelix.fileinstall.poll=5000
-Djava.endorsed.dirs=${com.sun.aas.installRoot}/modules/endorsed${path.separator}${com.sun.aas.installRoot
-Dosgi.shell.telnet.port=6666
-Dcom.ctc.wstx.returnNullForDefaultNamespace=true
-XX:+UnlockDiagnosticVMOptions
-Dcom.sun.enterprise.config.config_environment_factory_class=com.sun.enterprise.config.serverbeans.Appserv
-Djava.ext.dirs=${com.sun.aas.javaRoot}/lib/ext${path.separator}${com.sun.aas.javaRoot}/jre/lib/ext${path.sepa
-Djavax.xml.accessExternalSchema=all
-Dgosh.args=--nointeractive
-XX:MaxPermSize=192m
```

Figure 2.29: Clicking on the ADD JVM Option button

- Two empty rows will then be inserted, as depicted by Figure 2.30.



JVM Options

Manage JVM options for the server. Values containing one or more spaces must be enclosed in double quotes ("*value string*").

Configuration Name: server-config

```


-Djava.awt.headless=true
-Djava.security.policy=${com.sun.aas.instanceRoot}/config/server.policy
-Dfelix.fileinstall.disableConfigSave=false
-Dosgi.shell.telnet.maxconn=1
-Dfelix.fileinstall.poll=5000
-Djava.endorsed.dirs=${com.sun.aas.installRoot}/modules/endorsed${path.separator}${com.sun.aas.installRoot
-Dosgi.shell.telnet.port=6666
-Dcom.ctc.wstx.returnNullForDefaultNamespace=true
-XX:+UnlockDiagnosticVMOptions
-Dcom.sun.enterprise.config.config_environment_factory_class=com.sun.enterprise.config.serverbeans.Appserv
-Djava.ext.dirs=${com.sun.aas.javaRoot}/lib/ext${path.separator}${com.sun.aas.javaRoot}/jre/lib/ext${path.sepa
-Djavax.xml.accessExternalSchema=all
```

Figure 2.30: Two empty rows inserted in the JVM Options tab page

- Specify each of the following lines in each of the empty rows, as indicated by Figure 2.31:

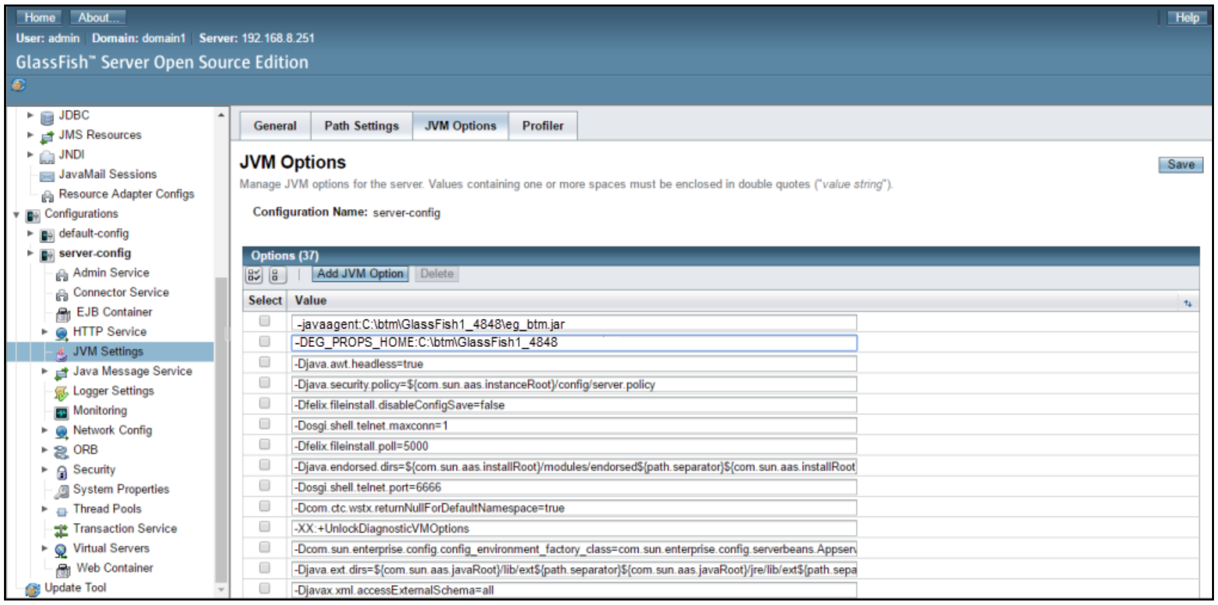
```
-javaagent:<Path_to_eg_btm.jar_file>
```

```
-DEG_PROPS_HOME:<Path_to_the_folder_containing_.props_files>
```

For instance, if the eg_ btm.jar file and .props files are extracted into the **C:\btm\GlassFish1_4848** directory, the above specification will be:

```
-javaagentC:\btm\GlassFish1_4848\eg_btm.jar
```

```
-DEG_PROPS_HOME:C:\btm\GlassFish1_4848
```



User: admin | Domain: domain1 | Server: 192.168.8.251

JVM Options

Manage JVM options for the server. Values containing one or more spaces must be enclosed in double quotes ("*value string*").

Configuration Name: server-config

Options (37)

Select	Value
☐	-javaagent:C:\btm\GlassFish1_4848\eg_btm.jar
☐	-DEG_PROPS_HOME:C:\btm\GlassFish1_4848
☐	-Djava.awt.headless=true
☐	-Djava.security.policy=${com.sun.aas.instanceRoot}/config/server.policy
☐	-Dfelix.fileinstall.disableConfigSave=false
☐	-Dosgi.shell.telnet.maxconn=1
☐	-Dfelix.fileinstall.poll=5000
☐	-Djava.endorsed.dirs=${com.sun.aas.installRoot}/modules/endorsed${path.separator}${com.sun.aas.installRoot
☐	-Dosgi.shell.telnet.port=6666
☐	-Dcom.ctc.wstx.returnNullForDefaultNamespace=true
☐	-XX:+UnlockDiagnosticVMOptions
☐	-Dcom.sun.enterprise.config.config_environment_factory_class=com.sun.enterprise.config.serverbeans.Appserv
☐	-Djava.ext.dirs=${com.sun.aas.javaRoot}/lib/ext${path.separator}${com.sun.aas.javaRoot}/jre/lib/ext${path.sepa
☐	-Djavax.xml.accessExternalSchema=all

Figure 2.31: Specifying the Java arguments for BTM-enabling the GlassFish server

- Finally, save the changes.
- Where multiple instances of GlassFish are monitored, make sure you perform steps 15-18above for each GlassFish server instance.

19. On the other hand, if you want to BTM-enable the GlassFish server by editing the start-up script of the GlassFish server instance, then follow the steps below:

- Open the start-up script and enter the following lines in it, as depicted by Figure 2.32.

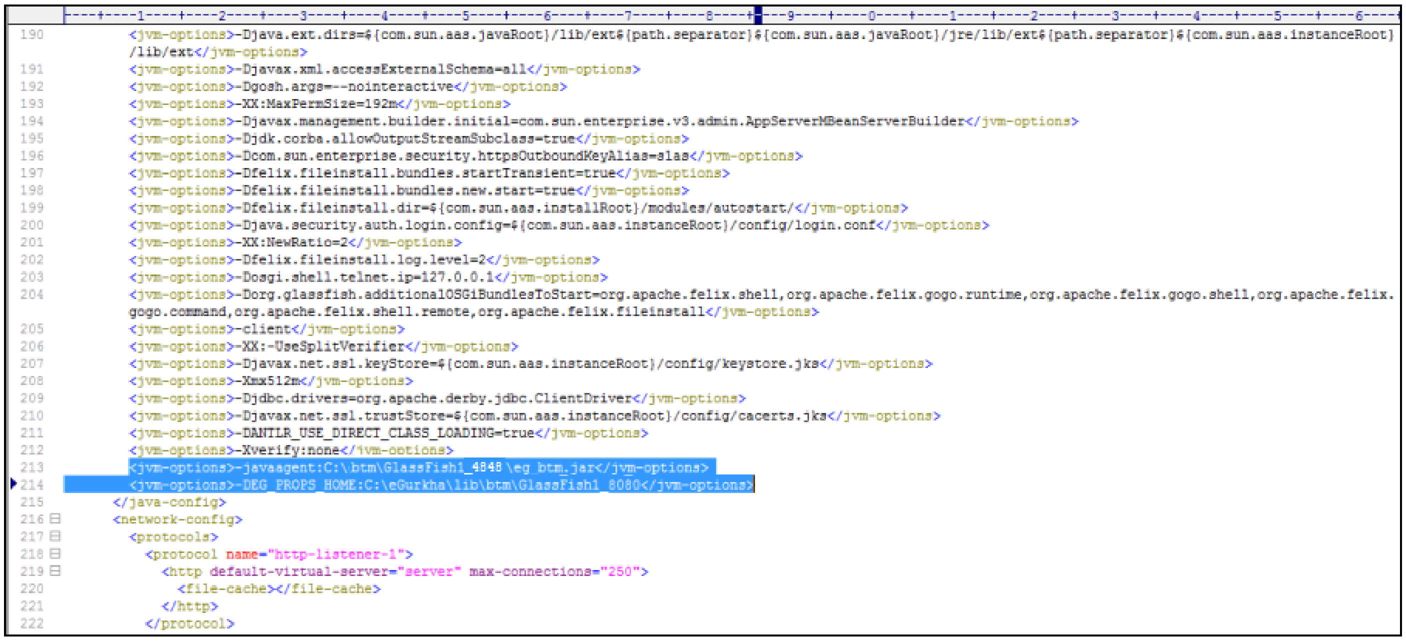
```
<jvm-options>-javaagent:<<Path_to_eg_btm.jar_file>></jvm-options>
```

```
<jvm- options>- DEG_ ROPS_ HOME=<<Path_ of_ the_ folder_ containing_.props_ files>></jvm-
options>
```

For instance, if the eg_btm.jar file and .props files are extracted into the **C:\btm\GlassFish1_ 4848** directory, the above specification will be:

```
<jvm-options>-javaagent:C:\btm\GlassFish1_4848\eg_btm.jar</jvm-options>
```

```
<jvm-options>-DEG_PROPS_HOME=C:\btm\GlassFish1_4848</jvm-options>
```

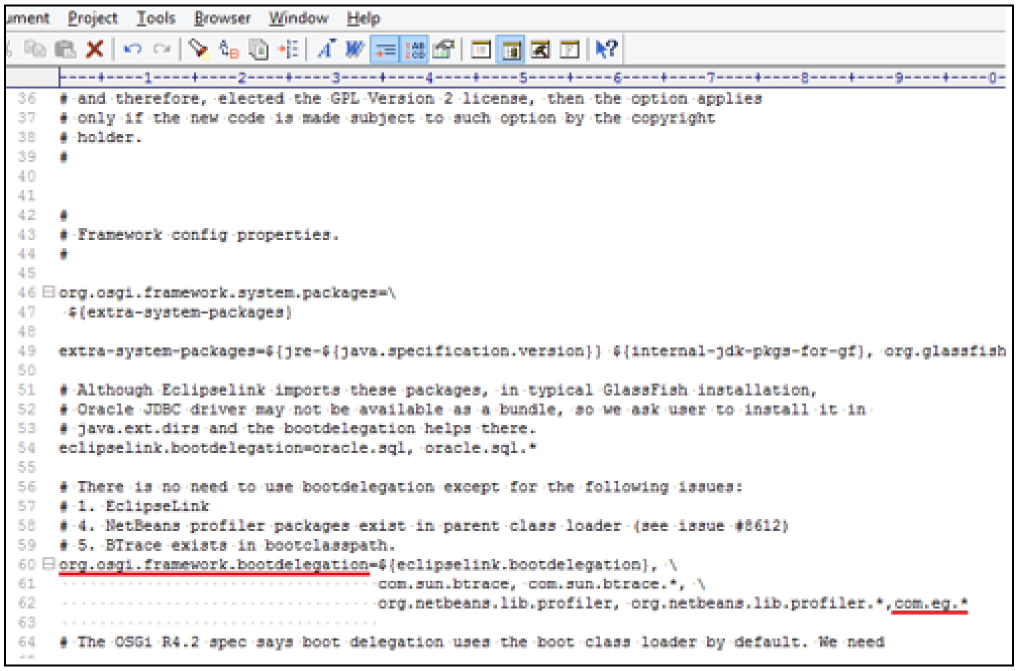


```
        <jvm-options>-Djava.ext.dirs=${com.sun.aas.javaRoot}/lib/ext${path.separator}${com.sun.aas.javaRoot}/jre/lib/ext${path.separator}${com.sun.aas.instanceRoot}
        /lib/ext</jvm-options>
        <jvm-options>-Djavax.xml.accessExternalSchema=all</jvm-options>
        <jvm-options>-Dgosh.args=--nointeractive</jvm-options>
        <jvm-options>-XX:MaxPermSize=192m</jvm-options>
        <jvm-options>-Djavax.management.builder.initial=com.sun.enterprise.v3.admin.AppServerMBeanServerBuilder</jvm-options>
        <jvm-options>-Djdk.corba.allowOutputStreamSubclass=true</jvm-options>
        <jvm-options>-Dcom.sun.enterprise.security.httpsOutboundKeyAlias=s1as</jvm-options>
        <jvm-options>-Dfelix.fileinstall.bundles.startTransient=true</jvm-options>
        <jvm-options>-Dfelix.fileinstall.bundles.new.start=true</jvm-options>
        <jvm-options>-Dfelix.fileinstall.dir=${com.sun.aas.installRoot}/modules/autostart/</jvm-options>
        <jvm-options>-Djava.security.auth.login.config=${com.sun.aas.instanceRoot}/config/login.conf</jvm-options>
        <jvm-options>-XX:NewRatio=2</jvm-options>
        <jvm-options>-Dfelix.fileinstall.log.level=2</jvm-options>
        <jvm-options>-Dosgi.shell.telnet.ip=127.0.0.1</jvm-options>
        <jvm-options>-Dorg.glassfish.additionalOSGiBundlesToStart=org.apache.felix.shell,org.apache.felix.gogo.runtime,org.apache.felix.gogo.shell,org.apache.felix.
        gogo.command,org.apache.felix.shell.remote,org.apache.felix.fileinstall</jvm-options>
        <jvm-options>-client</jvm-options>
        <jvm-options>-XX:-UseSplitVerifier</jvm-options>
        <jvm-options>-Djavax.net.ssl.keyStore=${com.sun.aas.instanceRoot}/config/keystore.jks</jvm-options>
        <jvm-options>-Xmx512m</jvm-options>
        <jvm-options>-Djdbc.drivers=org.apache.derby.jdbc.ClientDriver</jvm-options>
        <jvm-options>-Djavax.net.ssl.trustStore=${com.sun.aas.instanceRoot}/config/cacerts.jks</jvm-options>
        <jvm-options>-DANTLR_USE_DIRECT_CLASS_LOADING=true</jvm-options>
        <jvm-options>-Xverify:none</jvm-options>
        <jvm-options>-javaagent:C:\btm\GlassFish1_4848\eg_btm.jar</jvm-options>
        <jvm-options>-DEG_PROPS_HOME:C:\eGurkha\lib\btm\GlassFish1_8080</jvm-options>
      </java-config>
      <network-config>
        <protocols>
          <protocol name="http-listener-1">
            <http default-virtual-server="server" max-connections="250">
              <file-cache></file-cache>
            </http>
          </protocol>
```

Figure 2.32: Editing the start-up script of the GlassFish server instance to BTM-enable the instance

- Then, save the file.
- Where multiple instances of GlassFish are monitored, make sure you perform steps 15, 16, and 19 above for each GlassFish server instance.

20. Next, add the package com.eg in the boot delegation framework. If the Equinox OSGI container is used, then edit the <<GLASSFISH_ INSTALL_ DIR>>\glassfish\osgi\equinox\configuration\config.ini file to add the package "com.eg.*". If the Felix OSGI container is used, then edit the <<GLASSFISH_ INSTALL_ DIR>>\glassfish\osgi\felix\configuration\config.properties file to add the "com.eg.*". Figure 2.33 depicts how to add this package to the config file.



```
# and therefore, elected the GPL Version 2 license, then the option applies
# only if the new code is made subject to such option by the copyright
# holder.
#


#
# Framework config properties.
#

org.osgi.framework.system.packages=\
 ${extra-system-packages}

extra-system-packages=${jre-${java.specification.version}} ${internal-jdk-pkgs-for-gf}, org.glassfish

# Although Eclipselink imports these packages, in typical GlassFish installation,
# Oracle JDBC driver may not be available as a bundle, so we ask user to install it in
# java.ext.dirs and the bootdelegation helps there.
eclipselink.bootdelegation=oracle.sql, oracle.sql.*

# There is no need to use bootdelegation except for the following issues:
# 1. EclipseLink
# 4. NetBeans profiler packages exist in parent class loader (see issue #8612)
# 5. BTrace exists in bootclasspath.
org.osgi.framework.bootdelegation=${eclipselink.bootdelegation}, \
                                  com.sun.btrace, com.sun.btrace.*, \
                                  org.netbeans.lib.profiler, org.netbeans.lib.profiler.*,com.eg.*

# The OSGi R4.2 spec says boot delegation uses the boot class loader by default. We need
```

Figure 2.33: Adding com.eg to boot delegation framework

21. Finally, save the file and restart the Glassfish application server.

2.5.2 BTM-Enabling a GlassFish Server Running on a Unix Platform

If the target GlassFish server is running on a Unix operating system, then follow the steps below to BTM-enable that server:

1. Login to any system in your environment that supports a browser and has network access to the eG manager.
2. Open a browser on that system, connect to the eG manager, and login to the eG admin interface.
3. Manage the target GlassFish server as a separate component using the eG administrative interface. When managing, make a note of the **Nick name** and **Port number** that you provide.
4. If multiple GlassFish server instances are operating on a single host, and you want to monitor each of those instances, then you will have to manage each instance as a separate GlassFish server component using the eG administrative interface. When doing so, make a note of the **Nick name** and **Port number** using which you managed each instance.
5. Next, log out of the eG admin interface and the system.

6. Log into the target GlassFish server. Then, create a **btm** directory anywhere on the target server - say, **/opt/btm**. Under this directory, create a sub-folder. Make sure that you name this sub-folder in the following format: <*Managed_Component_NickName*>_<*Managed_Component_Port*>. For instance, if you have managed the GlassFish server using the nick name *GlassFish1* and the port number *4848*, the new directory under the **btm** directory should be named as *GlassFish1_4848*.
7. If you have managed multiple GlassFish server instances running on a single host, then you will have to create multiple sub-directories under the **btm** directory - one each for every instance. Each of these sub-directories should be named after the **Nick name** and **Port number** using which the corresponding instance has been managed in eG.
8. Once the new sub-directory(ies) is created, log out of the GlassFish server. Log back into the system you used in step 1 above. Open a browser on the system, connect to the eG manager, and login to the eG admin interface again.
9. Follow the Agents -> BTM Profiler Download menu sequence in the eG admin interface.
10. Figure 2.25 will appear listing the servers that can be instrumented for APM by eG. In this list, locate the GlassFish server that you want to BTM-enable. Once you locate the server, click the **Download** icon corresponding to that server to download the **APM Profiler Agent** to that server. If multiple GlassFish server instances on a single host are managed, then you will have to download the APM Profiler Agent separately for each of the managed instances.
11. Upon clicking the **Download** icon in Figure 2.25, a zip file named **javaagent_<Nick_name_of_GlassFish_server>_<Port_number_of_GlassFish_server** will get downloaded. For instance, if you have managed the GlassFish server using the nickname 'GlassFish1' and the port number '4848', then the name of the zip file will be *javaagent_GlassFish1_4848*. Where multiple GlassFish server instances have been managed, you will be downloading multiple zip files - one each for every instance. The names of these zip files will automatically carry the nick name and port number you assigned to the corresponding GlassFish server instance.
12. Next, using FTP tools like putty or WinSCP, transfer the downloaded zip file(s) to the corresponding sub-directory(ies) (of the **btm** directory), which you created on the GlassFish server in steps 6 and 7 above. For example, the zip file named *javaagent_GlassFish1_4848*, should be transferred to the**/opt/btm/GlassFish1_4848** directory on the target GlassFish server.
13. Log out of the system and log back into the GlassFish server.
14. Extract the contents of each zip file into the same sub-directory to which that zip file was copied.
15. Figure 2.26 depicts the extracted contents of the zip file.

16. From Figure 2.26, it is evident that the zip file contains an **eg_btm.jar** file and a few property files, namely - **btmOther.props** , **btmLogging.props** , **config.props** , **custom.props** , **exclude.props**, and **threshold.props** files.

17. Next, proceed to BTM-enable the GlassFish server/instance. For that, edit the **btmOther.props** file in the sub-directory (of the **btm** directory) that corresponds to that server/ instance. You will find the following lines in the file:

    ```
    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    # Below property is BTM Server Socket Port, through which eG Agent Communicates
    # Restart is required, if any changes in this property
    # Default port is "13931"
    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    #
    BTM_Port=13931
    #
    ```

    By default, the **BTMPort** parameter is set to 13931. If you want to enable eG Java BTM on a different port, then specify the same here. In this case, when configuring the **Java Business Transactions** test or the **Key Java Business Transactions** test for that application server, make sure you configure the **BTMPORT** parameter of the test with this port number.

    Note:

    When BTM-enabling multiple instances on the same server, make sure you configure a different **BTM Port** for each instance.

    Also, by default, the **Designated_Agent** parameter will be empty; do not disturb this default setting. In this case therefore, the eG Java BTM will treat the host from which the very first 'measure request' comes in as the **Designated_Agent**.

    Note:

    In case a specific **Designated_Agent** is not provided, and the eG Java BTM treats the host from which the very first 'measure request' comes in as the **Designated_Agent**, then if such a **Designated_Agent** is stopped or uninstalled for any reason, the eG Java BTM will wait for a maximum of 10 measure periods for that 'deemed' **Designated_Agent** to request for metrics. If no requests come in for 10 consecutive measure periods, then the eG Java BTM will begin responding to 'measure requests' coming in from any other eG agent.

18. Finally, save the **btmOther.props** file.

19. Then, proceed to configure the GlassFish server with the path to the .jar and .props files. For this, you need to edit the start-up script of the GlassFish server.
20. The first step towards this end is to open the start-up script. Then, insert the following lines in it:

```
<jvm-options>-javaagent:<<Path_to_the_eg_btm.jar_file>> </jvm-options>
```

```
<jvm- options>- DEG_ PROPS_ HOME=<<Path_ to_ the_ folder_ containing_.props_ files>></jvm-
options>
```

For instance, if the .props files had been copied to the **/opt/btm/GlassFish1_4848** directory, the above specification will be:

```
<jvm-options>-javaagent:/opt/btm/GlassFish1_4848/eg_btm.jar</jvm-options>
```

```
<jvm-options>-DEG_PROPS_HOME=/opt/btm/GlassFish1_4848</jvm-options>
```

21. In Unix environments, if the eG agent is deployed on the same host as the GlassFish server, then both the agent and the server will be running using different user privileges. In this situation, by default, the eG Java BTM logs will not be created. In order to create the same, insert the following entry after the -DEG_PROPS_HOME specification .

```
<jvm-options>-DEG_LOG_HOME=<<LogFile_Path>></jvm-options>
```

Before providing this specification, make sure you create a folder for BTM logs - say, **eGBTMLogs** - in any directory to which the target application server has access. Then, against, -DEG_LOG_HOME, provide the full path to the **eGBTMLogs** directory. Where multiple instances on the same server are to be BTM-enabled, you can use the same directory for writing log files of all instances.

For example, to create log files in the **/App001/eGBTMLogs** directory, the complete specification will be as follows:

```
<jvm-options>-javaagent:/opt/btm/GlassFish1_4848/eg_btm.jar</jvm-options>
```

```
<jvm-options>-DEG_PROPS_HOME=/opt/btm/GlassFish1_4848</jvm-options>
```

```
<jvm-options>-DEG_LOG_HOME=/App001/eGBTMLogs</jvm-options>
```

22. Then, save the file.
23. Where multiple instances of GlassFish are monitored, make sure you perform steps 17-22 above for each GlassFish server instance.
24. Next, add the package com.eg in the boot delegation framework. If the Equinox OSGI container is used, then edit the <<GLASSFISH_ INSTALL_

DIR>>/glassfish/osgi/equinox/configuration/config.ini file to add the package “com.eg.*". If the Felix OSGI container is used, then edit the <<GLASSFISH_ INSTALL_ DIR>>/glassfish/osgi/felix/configuration/config.properties file to add the "com.eg.*". In these files, look for the entry that begins with the *org.osgi.framework.bootdelegation*. Append the following string to that entry:

*,com.eg.**

25. Finally, save the file and restart the Glassfish application server.

2.6 Installing eG Java BTM on JBoss EAP

The steps for deploying the eG Java BTM on a JBoss EAP server will differ based on the platform on which the target JBoss EAP server is running - whether on a Windows platform or a Unix platform.

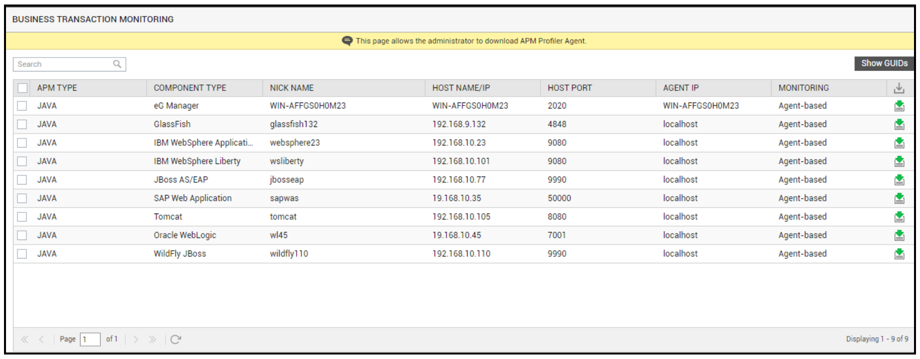
2.6.1 BTM-Enabling a JBoss EAP Server Running on a Windows Platform

If the JBoss EAP server is running on a Windows operating system, then follow the steps below to BTM-enable that server:

1. Login to the JBoss EAP server.
2. Open a browser on the server, connect to the eG manager, and login to the eG admin interface.
3. Manage the JBoss EAP server as a separate component using the eG administrative interface. When managing, make a note of the **Nick name** and **Port number** that you provide.
4. If multiple JBoss EAP server instances are operating on a single host, and you want to BTM-enable all the instances, then you will have to manage each instance as a separate component using the eG administrative interface. When doing so, make a note of the **Nick name** and **Port number** using which you managed each instance.
5. Next, log out of the eG admin interface. Then, create a **btm** directory anywhere on the JBoss EAP server - say, **C:\btm**. Under this directory, create a sub-folder. Make sure that you name this sub- folder in the following format: *<Managed_ Component_ NickName>_ <Managed_ Component_Port>*. For instance, if you have managed the JBoss EAP server using the nick name *JBoss1* and the port number *9990*, the new directory under the **btm** directory should be named as *JBoss1_9990*.
6. If you have managed multiple JBoss EAP server instances running on a single host, then you will have to create multiple sub-directories under the **btm** directory - one each for every instance. Each of these sub-directories should be named after the **Nick name** and **Port number** using

which the corresponding instance has been managed in eG.

7. Once the new sub-directory(ies) is created, open a browser on the JBoss EAP server, connect to the eG manager, and login to the eG admin interface again.
8. Follow the Agents -> BTM Profiler Download menu sequence in the eG admin interface.
9. Figure 2.34 will appear listing the servers that can be instrumented for APM by eG. In this list, locate the JBoss EAP server that you want to BTM-enable. Once you locate the server, click the **Download** icon corresponding to that server to download the **APM Profiler Agent** to that server. If multiple JBoss EAP server instances on a single host are managed, then you will have to download the APM Profiler Agent separately for each of the managed instances.

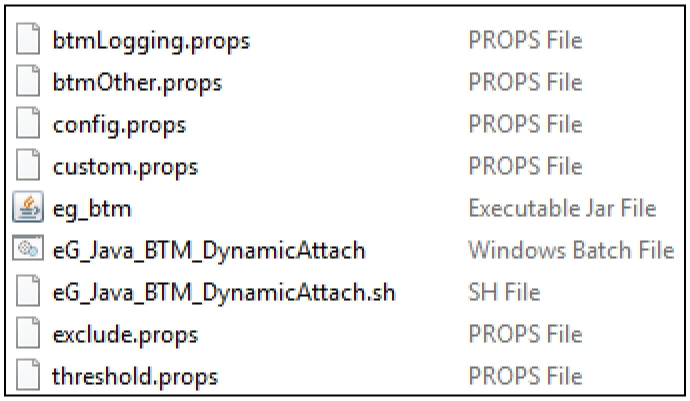

BUSINESS TRANSACTION MONITORING

This page allows the administrator to download APM Profiler Agent.

APM TYPE	COMPONENT TYPE	NICK NAME	HOST NAME/IP	HOST PORT	AGENT IP	MONITORING
JAVA	eG Manager	WIN-AFFGS0H0M23	WIN-AFFGS0H0M23	2020	WIN-AFFGS0H0M23	Agent-based
JAVA	GlassFish	glassfish132	192.168.9.132	4848	localhost	Agent-based
JAVA	IBM WebSphere Applicati...	websphere23	192.168.10.23	9080	localhost	Agent-based
JAVA	IBM WebSphere Liberty	wsliberty	192.168.10.101	9080	localhost	Agent-based
JAVA	JBoss AS/EAP	jbosseap	192.168.10.77	9990	localhost	Agent-based
JAVA	SAP Web Application	sapwas	19.168.10.35	50000	localhost	Agent-based
JAVA	Tomcat	tomcat	192.168.10.105	8080	localhost	Agent-based
JAVA	Oracle WebLogic	wl45	19.168.10.45	7001	localhost	Agent-based
JAVA	WildFly JBoss	wildfly110	192.168.10.110	9990	localhost	Agent-based

Figure 2.34: Downloading the APM Profiler Agent for the JBoss EAP server

10. Upon clicking the **Download** icon in Figure 2.34, a zip file named **javaagent_<Nick_name_of_ JBossEAP_ server>_ <Port_ number_ of_ JBossEAP_ server** will get downloaded. For instance, if you have managed the JBoss EAP server using the nickname 'JBoss1' and the port number '9990', then the name of the zip file will be *javaagent_JBoss1_9990*. Where multiple JBoss EAP server instances have been managed, you will be downloading multiple zip files - one each for every instance. The names of these zip files will automatically carry the nick name and port number you assigned to the corresponding server instance.
11. Copy the downloaded zip file(s) to the corresponding sub-directory(ies) of the **btm** directory (see steps 5 and 6 above). For example, the zip file named *javaagent_JBoss1_9990*, should be copied to the **C:\btm\JBoss1_9990** directory .
12. Extract the contents of each zip file into the same sub-directory to which that zip file was copied.
13. Figure 2.35 depicts the extracted contents of the zip file.



btmLogging.props	PROPS File
btmOther.props	PROPS File
config.props	PROPS File
custom.props	PROPS File
eg_btm	Executable Jar File
eG_Java_BTM_DynamicAttach	Windows Batch File
eG_Java_BTM_DynamicAttach.sh	SH File
exclude.props	PROPS File
threshold.props	PROPS File

Figure 2.35: Contents of the APM Profiler Agent zip

14. From Figure 2.35, it is evident that the zip file contains an **eg_btm.jar** file and a few property files, namely - **btmOther.props** , **btmLogging.props** , **config.props** , **custom.props** , **exclude.props**, and **threshold.props** files.

15. Next, proceed to BTM- enable the JBoss EAP server/instance. For that, edit the **btmOther.props** file in the sub-directory (of the **btm** directory) that corresponds to that JBoss EAP server/ instance. You will find the following lines in the file:

```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Below property is BTM Server Socket Port, through which eG Agent Communicates
# Restart is required, if any changes in this property
# Default port is "13931"
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#
BTM_Port=13931
#
```

By default, the **BTMPort** parameter is set to 13931. If you want to enable eG Java BTM on a different port, then specify the same here. In this case, when configuring the **Java Business Transactions** test or the **Key Java Business Transactions** test for that application server, make sure you configure the **BTM** port parameter of the test with this port number.

Note:

When BTM-enabling multiple instances on the same server, make sure you configure a different **BTM Port** for each instance.

Also, by default, the **Designated_Agent** parameter will be empty; do not disturb this default setting. In this case therefore, the eG Java BTM will treat the host from which the very first 'measure request' comes in as the **Designated_Agent**.

Note:

In case a specific **Designated_Agent** is not provided, and the eG Java BTM treats the host from which the very first 'measure request' comes in as the **Designated_Agent**, then if such a **Designated_Agent** is stopped or uninstalled for any reason, the eG Java BTM will wait for a maximum of 10 measure periods for that 'deemed' **Designated_Agent** to request for metrics. If no requests come in for 10 consecutive measure periods, then the eG Java BTM will begin responding to 'measure requests' coming in from any other eG agent.

16. Finally, save the **btmOther.props** file.

17. Then, you need to configure the JBoss EAP server with the path to the **eg_btm.jar** and **.props** files. To achieve this, you need to edit the start-up script of the JBoss EAP server. The first step towards that is to open the start-up script.

18. Then, in the file, enter the following lines, as depicted by Figure 2.36.

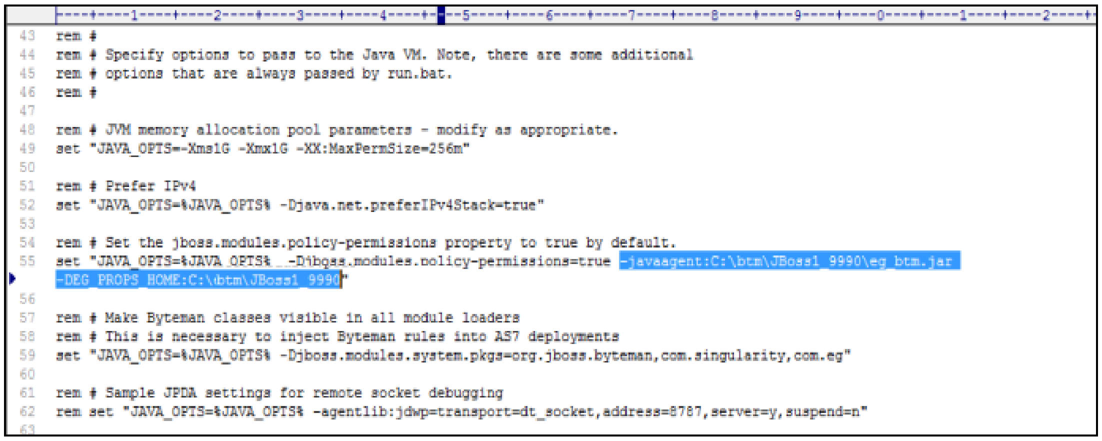
```
-javaagent:<<PATH TO THE eg_btm.jar FILE>>
```

```
-DEG_PROPS_HOME=<<PATH OF THE LOCAL FOLDER CONTAINING THE .PROPS FILES>>
```

For instance, if the eg_btm.jar file and .props files are extracted into the **C:\btm\JBoss1_9990** directory, the above specification will be:

```
-javaagent:C:\btm\JBoss1_9990\eg_btm.jar
```

```
-DEG_PROPS_HOME:C:\btm\JBoss1_9990
```

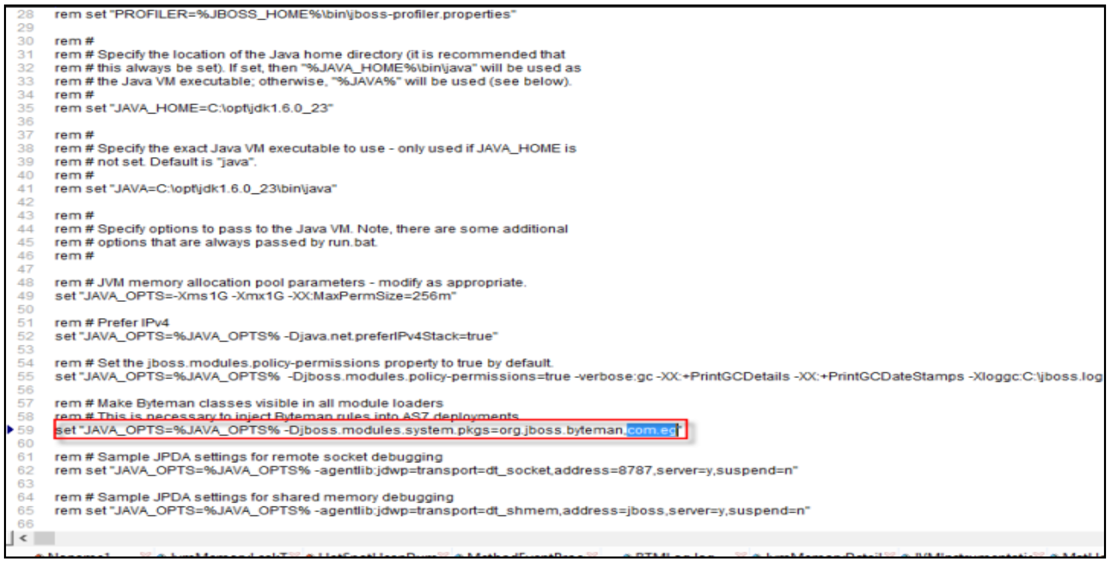


```
rem #
rem # Specify options to pass to the Java VM. Note, there are some additional
rem # options that are always passed by run.bat.
rem #

rem # JVM memory allocation pool parameters - modify as appropriate.
set "JAVA_OPTS=-Xms1G -Xmx1G -XX:MaxPermSize=256m"

rem # Prefer IPv4
set "JAVA_OPTS=%JAVA_OPTS% -Djava.net.preferIPv4Stack=true"

rem # Set the jboss.modules.policy-permissions property to true by default.
set "JAVA_OPTS=%JAVA_OPTS%  -Djboss.modules.policy-permissions=true -javaagent:C:\btm\JBoss1_9990\eg_btm.jar
-DEG_PROPS_HOME:C:\btm\JBoss1_9990"

rem # Make Byteman classes visible in all module loaders
rem # This is necessary to inject Byteman rules into AS7 deployments
set "JAVA_OPTS=%JAVA_OPTS% -Djboss.modules.system.pkgs=org.jboss.byteman,com.singularity,com.eg"

rem # Sample JPDA settings for remote socket debugging
rem set "JAVA_OPTS=%JAVA_OPTS% -agentlib:jdwp=transport=dt_socket,address=8787,server=y,suspend=n"
```

Figure 2.36: Editing the start-up script to BTM-enable a JBoss EAP server running on Windows

19. Where multiple instances of JBoss EAP are monitored, make sure you perform steps 15-18 above for each JBoss EAP server instance.

20. Next, if the JBoss server is running in domain mode, open the **domain.conf** file in **JBoss_Home\bin** directory. If the JBoss server is running in standalone mode, open the **standalone.conf** file in the **JBoss_Home\bin** directory.



```
rem set "PROFILER=%JBOSS_HOME%\bin\jboss-profiler.properties"

rem #
rem # Specify the location of the Java home directory (it is recommended that
rem # this always be set). If set, then "%JAVA_HOME%\bin\java" will be used as
rem # the Java VM executable; otherwise, "%JAVA%" will be used (see below).
rem #
rem set "JAVA_HOME=C:\opt\jdk1.6.0_23"

rem #
rem # Specify the exact Java VM executable to use - only used if JAVA_HOME is
rem # not set. Default is "java".
rem #
rem set "JAVA=C:\opt\jdk1.6.0_23\bin\java"

rem #
rem # Specify options to pass to the Java VM. Note, there are some additional
rem # options that are always passed by run.bat.
rem #

rem # JVM memory allocation pool parameters - modify as appropriate.
set "JAVA_OPTS=-Xms1G -Xmx1G -XX:MaxPermSize=256m"

rem # Prefer IPv4
set "JAVA_OPTS=%JAVA_OPTS% -Djava.net.preferIPv4Stack=true"

rem # Set the jboss.modules.policy-permissions property to true by default.
set "JAVA_OPTS=%JAVA_OPTS%  -Djboss.modules.policy-permissions=true -verbose:gc -XX:+PrintGCDetails -XX:+PrintGCDateStamps -Xloggc:C:\jboss.log

rem # Make Byteman classes visible in all module loaders
rem # This is necessary to inject Byteman rules into AS7 deployments
set "JAVA_OPTS=%JAVA_OPTS% -Djboss.modules.system.pkgs=org.jboss.byteman,com.eg"

rem # Sample JPDA settings for remote socket debugging
rem set "JAVA_OPTS=%JAVA_OPTS% -agentlib:jdwp=transport=dt_socket,address=8787,server=y,suspend=n"

rem # Sample JPDA settings for shared memory debugging
rem set "JAVA_OPTS=%JAVA_OPTS% -agentlib:jdwp=transport=dt_shmem,address=jboss,server=y,suspend=n"
```

Figure 2.37: Editing the domain.conf file or standalone.conf file

21. Append **",com.eg"** to the following line in the file, as depicted by Figure 2.37:

```
set "JAVA_OPTS=%JAVA_OPTS% -Djboss.modules.system.pkgs="org.jboss.byteman"
```

22. Finally, save the file. and restart the JBoss EAP server.

2.6.2 BTM-Enabling a JBoss EAP Server Running on a Unix Platform

If the target JBoss EAP server is running on a Unix operating system, then follow the steps below to BTM-enable that server:

1. Login to any system in your environment that supports a browser and has network access to the eG manager.
2. Open a browser on that system, connect to the eG manager, and login to the eG admin interface.
3. Manage the target JBoss EAP server as a separate component using the eG administrative interface. When managing, make a note of the **Nick name** and **Port number** that you provide.
4. If multiple JBoss EAP server instances are operating on a single host, and you want to monitor each of those instances, then you will have to manage each instance as a separate JBoss EAP server component using the eG administrative interface. When doing so, make a note of the **Nick name** and **Port number** using which you managed each instance.
5. Next, log out of the eG admin interface and the system.
6. Log into the target JBoss EAP server. Then, create a **btm** directory anywhere on the target server - say, **/opt/btm**. Under this directory, create a sub-folder. Make sure that you name this sub- folder in the following format: *<Managed_ Component_ NickName>_ <Managed_ Component_Port>*. For instance, if you have managed the JBoss EAP server using the nick name *JBoss1* and the port number *9990*, the new directory under the **btm** directory should be named as *JBoss1_9990*.
7. If you have managed multiple JBoss server instances running on a single host, then you will have to create multiple sub-directories under the **btm** directory - one each for every instance. Each of these sub-directories should be named after the **Nick name** and **Port number** using which the corresponding instance has been managed in eG.
8. Once the new sub-directory(ies) is created, log out of the JBoss EAP server. Log back into the system you used in step 1 above. Open a browser on the system, connect to the eG manager, and login to the eG admin interface again.
9. Follow the Agents -> BTM Profiler Download menu sequence in the eG admin interface.
10. Figure 2.34 will appear listing the servers that can be instrumented for APM by eG. In this list, locate the JBoss EAP server that you want to BTM-enable. Once you locate the server, click the

Download icon corresponding to that server to download the **APM Profiler Agent** to that server. If multiple JBoss EAP server instances on a single host are managed, then you will have to download the APM Profiler Agent separately for each of the managed instances.

11. Upon clicking the **Download** icon in Figure 2.34, a zip file named **javaagent_<Nick_name_of_ JBossEAP_ server>_ <Port_ number_ of_ JBossEAP_ server** will get downloaded. For instance, if you have managed the JBoss EAP server using the nickname 'JBoss1' and the port number '9990', then the name of the zip file will be *javaagent_JBoss1_9990*. Where multiple JBoss EAP server instances have been managed, you will be downloading multiple zip files - one each for every instance. The names of these zip files will automatically carry the nick name and port number you assigned to the corresponding JBoss EAP server instance.
12. Next, using FTP tools like putty or WinSCP, transfer the downloaded zip file (s) to the corresponding sub-directory(ies) (of the **btm** directory), which you created on the JBoss EAP server in steps 6 and 7 above. For example, the zip file named *javaagent_JBoss1_9990*, should be transferred to the**/opt/btm/JBoss1_9990** directory on the target JBoss EAP server.
13. Log out of the system and log back into the JBoss EAP server.
14. Extract the contents of each zip file into the same sub-directory to which that zip file was copied.
15. Figure 2.35 depicts the extracted contents of the zip file.
16. From Figure 2.35, it is evident that the zip file contains an **eg_btm.jar** file and a few property files, namely - **btmOther.props** , **btmLogging.props** , **config.props** , **custom.props** , **exclude.props**, and **threshold.props** files.
17. Next, proceed to BTM- enable the JBoss EAP server/instance. For that, edit the **btmOther.props** file in the sub-directory (of the **btm** directory) that corresponds to that server/ instance. You will find the following lines in the file:

```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Below property is BTM Server Socket Port, through which eG Agent Communicates
# Restart is required, if any changes in this property
# Default port is "13931"
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#
BTM_Port=13931
#
```

By default, the **BTMPort** parameter is set to 13931. If you want to enable eG Java BTM on a different port, then specify the same here. In this case, when configuring the **Java Business Transactions** test or the **Key Java Business Transactions** test for that application server, make sure you configure the **BTMPORT** parameter of the test with this port number.

Note:

When BTM-enabling multiple instances on the same server, make sure you configure a different **BTM Port** for each instance.

Also, by default, the **Designated_Agent** parameter will be empty; do not disturb this default setting. In this case therefore, the eG Java BTM will treat the host from which the very first 'measure request' comes in as the **Designated_Agent**.

Note:

In case a specific **Designated_Agent** is not provided, and the eG Java BTM treats the host from which the very first 'measure request' comes in as the **Designated_Agent**, then if such a **Designated_Agent** is stopped or uninstalled for any reason, the eG Java BTM will wait for a maximum of 10 measure periods for that 'deemed' **Designated_Agent** to request for metrics. If no requests come in for 10 consecutive measure periods, then the eG Java BTM will begin responding to 'measure requests' coming in from any other eG agent.

18. Finally, save the **btmOther.props** file.

19. Then, you need to configure the JBoss EAP server with the path to the **eg_btm.jar** and **.props** files. To achieve this, you need to edit the start-up script of the server. The first step towards that is to open the start-up script.

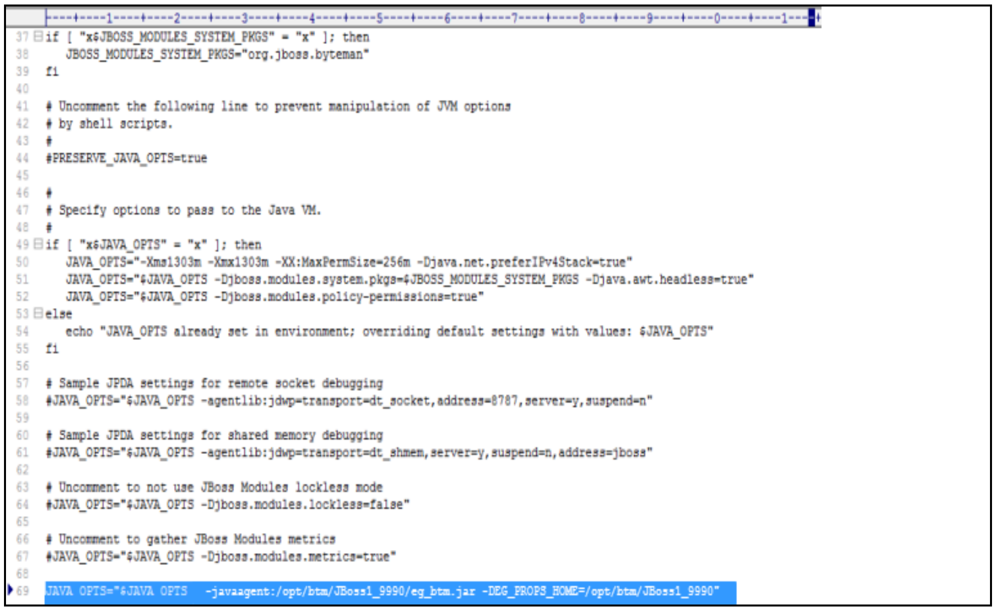
20. Then, in the file, enter the following lines, as depicted by Figure 2.38.

```
JAVA_OPTS="$JAVA_OPTS -javaagent:<<PATH TO the eg_btm.jar>> -DEG_PROPS_HOME=<<PATH TO
LOCAL FOLDER CONTAINING THE
```

```
.PROPS FILES>>"
```

For instance, if the eg_btm.jar file and the .props files are extracted into the **/opt/btm/JBoss1_ 9990** directory, the above specification will be:

```
JAVA_OPTS="$JAVA_OPTS -javaagent:/opt/btm/JBoss1_9990/eg_btm.jar -DEG_PROPS_
HOME=/opt/btm/JBoss1_9990"
```



```
if [ "x$JBOSS_MODULES_SYSTEM_PKGS" = "x" ]; then
   JBOSS_MODULES_SYSTEM_PKGS="org.jboss.byteman"
fi

# Uncomment the following line to prevent manipulation of JVM options
# by shell scripts.
#
#PRESERVE_JAVA_OPTS=true

#
# Specify options to pass to the Java VM.
#
if [ "x$JAVA_OPTS" = "x" ]; then
   JAVA_OPTS="-Xms1303m -Xmx1303m -XX:MaxPermSize=256m -Djava.net.preferIPv4Stack=true"
   JAVA_OPTS="$JAVA_OPTS -Djboss.modules.system.pkgs=$JBOSS_MODULES_SYSTEM_PKGS -Djava.awt.headless=true"
   JAVA_OPTS="$JAVA_OPTS -Djboss.modules.policy-permissions=true"
else
   echo "JAVA_OPTS already set in environment; overriding default settings with values: $JAVA_OPTS"
fi

# Sample JPDA settings for remote socket debugging
#JAVA_OPTS="$JAVA_OPTS -agentlib:jdwp=transport=dt_socket,address=8787,server=y,suspend=n"

# Sample JPDA settings for shared memory debugging
#JAVA_OPTS="$JAVA_OPTS -agentlib:jdwp=transport=dt_shmem,server=y,suspend=n,address=jboss"

# Uncomment to not use JBoss Modules lockless mode
#JAVA_OPTS="$JAVA_OPTS -Djboss.modules.lockless=false"

# Uncomment to gather JBoss Modules metrics
#JAVA_OPTS="$JAVA_OPTS -Djboss.modules.metrics=true"

JAVA_OPTS="$JAVA_OPTS  -javaagent:/opt/btm/JBoss1_9990/eg_btm.jar -DEG_PROPS_HOME=/opt/btm/JBoss1_9990"
```

Figure 2.38: Editing the start-up script to BTM-enable a JBoss EAP server on Unix that is monitored in an agent-based manner

21. In Unix environments, if the eG agent is deployed on the same host as the JBoss EAP server, then both the agent and the server will be running using different user privileges. In this situation, by default, the eG Java BTM logs will not be created. In order to create the same, insert the following entry after the -DEG_PROPS_HOME specification, but before the closing quotes.

    ```
    -DEG_LOG_HOME=<<LogFile_Path>>
    ```

    Before providing this specification, make sure you create a folder for BTM logs - say, **eGBTMLogs** - in any directory to which the target application server has access. Then, against, -DEG_LOG_HOME, provide the full path to the **eGBTMLogs** directory. Where multiple instances on the same server are to be BTM-enabled, you can use the same directory for writing log files of all instances.

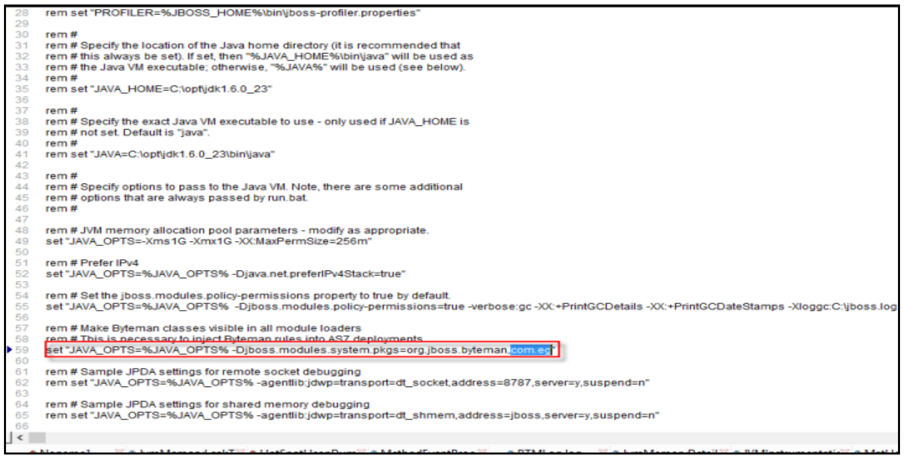
    For example, to create log files in the **/App001/eGBTMLogs** directory, the complete specification will be as follows:

    ```
    JAVA_OPTS="$JAVA_OPTS -javaagent:/opt/btm/JBoss1_9990/eg_btm.jar -DEG_PROPS_
    HOME=/opt/btm/JBoss1_9990 -DEG_LOG_HOME=/App001/eGBTMLogs"
    ```

22. Then, save the file.

23. Where multiple instances of JBoss EAP are monitored, make sure you perform steps 17-22 above for each JBoss EAP server instance.

24. Next, if the JBoss server is running in domain mode, open the **domain.conf** file in **JBoss_ Home/bin** directory. If the JBoss server is running in standalone mode, open the **standalone.conf** file in the **JBoss_Home/bin** directory.



```
rem set "PROFILER=%JBOSS_HOME%\bin\jboss-profiler.properties"

rem #
rem # Specify the location of the Java home directory (it is recommended that
rem # this always be set). If set, then "%JAVA_HOME%\bin\java" will be used as
rem # the Java VM executable; otherwise, "%JAVA%" will be used (see below).
rem #
rem set "JAVA_HOME=C:\opt\jdk1.6.0_23"

rem #
rem # Specify the exact Java VM executable to use - only used if JAVA_HOME is
rem # not set. Default is "java".
rem #
rem set "JAVA=C:\opt\jdk1.6.0_23\bin\java"

rem #
rem # Specify options to pass to the Java VM. Note, there are some additional
rem # options that are always passed by run.bat.
rem #

rem # JVM memory allocation pool parameters - modify as appropriate.
set "JAVA_OPTS=-Xms1G -Xmx1G -XX:MaxPermSize=256m"

rem # Prefer IPv4
set "JAVA_OPTS=%JAVA_OPTS% -Djava.net.preferIPv4Stack=true"

rem # Set the jboss.modules.policy-permissions property to true by default.
set "JAVA_OPTS=%JAVA_OPTS%  -Djboss.modules.policy-permissions=true -verbose:gc -XX:+PrintGCDetails -XX:+PrintGCDateStamps -Xloggc:C:\jboss.log

rem # Make Byteman classes visible in all module loaders
rem # This is necessary to inject Byteman rules into AS7 deployments
set "JAVA_OPTS=%JAVA_OPTS% -Djboss.modules.system.pkgs=org.jboss.byteman,com.eg"

rem # Sample JPDA settings for remote socket debugging
rem set "JAVA_OPTS=%JAVA_OPTS% -agentlib:jdwp=transport=dt_socket,address=8787,server=y,suspend=n"

rem # Sample JPDA settings for shared memory debugging
rem set "JAVA_OPTS=%JAVA_OPTS% -agentlib:jdwp=transport=dt_shmem,address=jboss,server=y,suspend=n"
```

Figure 2.39: Editing the domain.conf file or standalone.conf file

25. Append **",com.eg"** to the following line in the file, as depicted by Figure 2.37:

```
set "JAVA_OPTS=%JAVA_OPTS% -Djboss.modules.system.pkgs="org.jboss.byteman"
```

26. Finally, save the file and restart the JBoss EAP server instance.

2.7 Installing eG Java BTM on JBoss WildFly

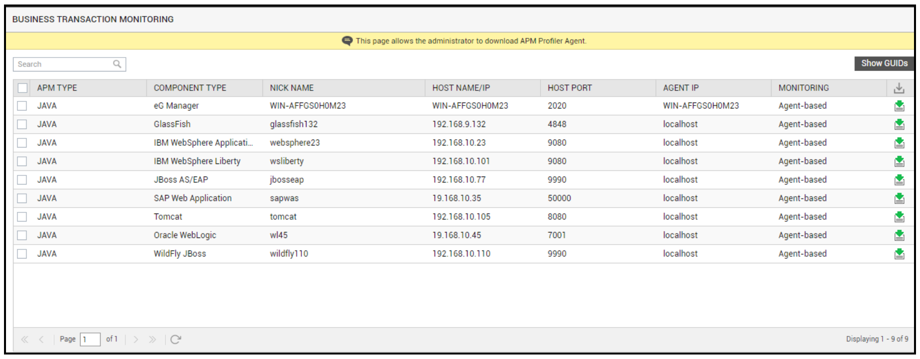
The steps for deploying the eG Java BTM on a JBoss WildFly server will differ based on the platform on which the target JBoss WildFly server is running - whether on a Windows platform or a Unix platform.

2.7.1 BTM-Enabling a JBoss WildFly Server Running on a Windows Platform

If the JBoss WildFly server is running on a Windows operating system, then follow the steps below to BTM-enable that server:

1. Login to the JBoss WildFly server.
2. Open a browser on the server, connect to the eG manager, and login to the eG admin interface.

3. Manage the JBoss WildFly server as a separate component using the eG administrative interface. When managing, make a note of the **Nick name** and **Port number** that you provide.
4. If multiple JBoss WildFly server instances are operating on a single host, and you want to BTM-enable all the instances, then you will have to manage each instance as a separate component using the eG administrative interface. When doing so, make a note of the **Nick name** and **Port number** using which you managed each instance.
5. Next, log out of the eG admin interface. Then, create a **btm** directory anywhere on the JBoss WildFly server - say, **C:\btm**. Under this directory, create a sub-folder. Make sure that you name this sub- folder in the following format: *<Managed_ Component_ NickName>_ <Managed_ Component_Port>*. For instance, if you have managed the JBoss WildFly server using the nick name *WildFly1* and the port number *9990*, the new directory under the **btm** directory should be named as *WildFly1_9990*.
6. If you have managed multiple JBoss WildFly server instances running on a single host, then you will have to create multiple sub-directories under the **btm** directory- one each for every instance. Each of these sub-directories should be named after the **Nick name** and **Port number** using which the corresponding instance has been managed in eG.
7. Once the new sub-directory(ies) is created, open a browser on the JBoss WildFly server, connect to the eG manager, and login to the eG admin interface again.
8. Follow the Agents -> BTM Profiler Download menu sequence in the eG admin interface.
9. Figure 2.40 will appear listing the servers that can be instrumented for APM by eG. In this list, locate the JBoss WildFly server that you want to BTM-enable. Once you locate the server, click the **Download** icon corresponding to that server to download the **APM Profiler Agent** to that server. If multiple JBoss WildFly server instances on a single host are managed, then you will have to download the APM Profiler Agent separately for each of the managed instances.

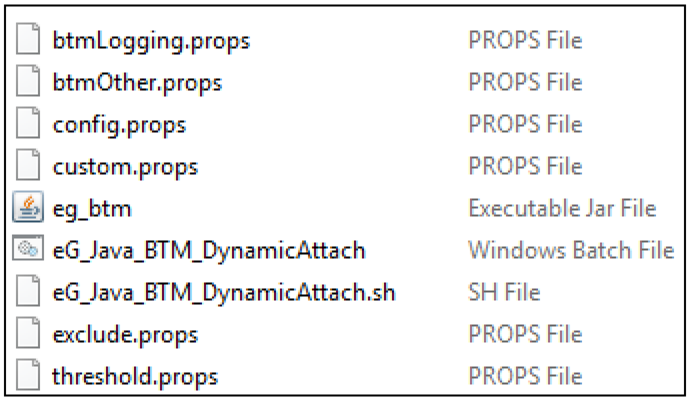

BUSINESS TRANSACTION MONITORING

This page allows the administrator to download APM Profiler Agent.

APM TYPE	COMPONENT TYPE	NICK NAME	HOST NAME/IP	HOST PORT	AGENT IP	MONITORING	
JAVA	eG Manager	WIN-AFFGS0H0M23	WIN-AFFGS0H0M23	2020	WIN-AFFGS0H0M23	Agent-based	
JAVA	GlassFish	glassfish132	192.168.9.132	4848	localhost	Agent-based	
JAVA	IBM WebSphere Applicati...	websphere23	192.168.10.23	9080	localhost	Agent-based	
JAVA	IBM WebSphere Liberty	wsliberty	192.168.10.101	9080	localhost	Agent-based	
JAVA	JBoss AS/EAP	jbosseap	192.168.10.77	9990	localhost	Agent-based	
JAVA	SAP Web Application	sapwas	19.168.10.35	50000	localhost	Agent-based	
JAVA	Tomcat	tomcat	192.168.10.105	8080	localhost	Agent-based	
JAVA	Oracle WebLogic	wl45	19.168.10.45	7001	localhost	Agent-based	
JAVA	WildFly JBoss	wildfly110	192.168.10.110	9990	localhost	Agent-based	

Figure 2.40: Downloading the APM Profiler Agent for the JBoss WildFly server

10. Upon clicking the **Download** icon in Figure 2.40, a zip file named **javaagent_<Nick_name_of_WildFly_server>_<Port_number_of_WildFly_server** will get downloaded. For instance, if you have managed the JBoss WildFly server using the nickname 'WildFly1' and the port number '9990', then the name of the zip file will be *javaagent_WildFly1_9990*. Where multiple JBoss WildFly server instances have been managed, you will be downloading multiple zip files - one each for every instance. The names of these zip files will automatically carry the nick name and port number you assigned to the corresponding server instance.
11. Copy the downloaded zip file(s) to the corresponding sub-directory(ies) of the **btm** directory (see steps 5 and 6 above). For example, the zip file named *javaagent_WildFly1_9990*, should be copied to the **C:\btm\WildFly1_9990** directory .
12. Extract the contents of each zip file into the same sub-directory to which that zip file was copied.
13. Figure 2.41 depicts the extracted contents of the zip file.



btmLogging.props	PROPS File
btmOther.props	PROPS File
config.props	PROPS File
custom.props	PROPS File
eg_btm	Executable Jar File
eG_Java_BTM_DynamicAttach	Windows Batch File
eG_Java_BTM_DynamicAttach.sh	SH File
exclude.props	PROPS File
threshold.props	PROPS File

Figure 2.41: Contents of the APM Profiler Agent zip

14. From Figure 2.41, it is evident that the zip file contains an **eg_btm.jar** file and a few property files, namely - **btmOther.props** , **btmLogging.props** , **config.props** , **custom.props** , **exclude.props**, and **threshold.props** files.
15. Next, proceed to BTM- enable the JBoss WildFly server/instance. For that, edit the **btmOther.props** file in the sub-directory (of the **btm** directory) that corresponds to that JBoss WildFly server/ instance. You will find the following lines in the file:

```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Below property is BTM Server Socket Port, through which eG Agent Communicates
# Restart is required, if any changes in this property
# Default port is "13931"
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
#
BTM_Port=13931
#
```

By default, the **BTMPort** parameter is set to 13931. If you want to enable eG Java BTM on a different port, then specify the same here. In this case, when configuring the **Java Business Transactions** test or the **Key Java Business Transactions** test for that application server, make sure you configure the **BTM** port parameter of the test with this port number.

Note:

When BTM-enabling multiple instances on the same server, make sure you configure a different **BTM Port** for each instance.

Also, by default, the **Designated_Agent** parameter will be empty; do not disturb this default setting. In this case therefore, the eG Java BTM will treat the host from which the very first 'measure request' comes in as the **Designated_Agent**.

Note:

In case a specific **Designated_Agent** is not provided, and the eG Java BTM treats the host from which the very first 'measure request' comes in as the **Designated_Agent**, then if such a **Designated_Agent** is stopped or uninstalled for any reason, the eG Java BTM will wait for a maximum of 10 measure periods for that 'deemed' **Designated_Agent** to request for metrics. If no requests come in for 10 consecutive measure periods, then the eG Java BTM will begin responding to 'measure requests' coming in from any other eG agent.

16. Finally, save the **btmOther.props** file.

17. Then, you need to configure the JBoss WildFly server with the path to the **eg_btm.jar** and **.props** files. To achieve this, you need to edit the start-up script of the JBoss WildFly server. The first step towards that is to open the start-up script.

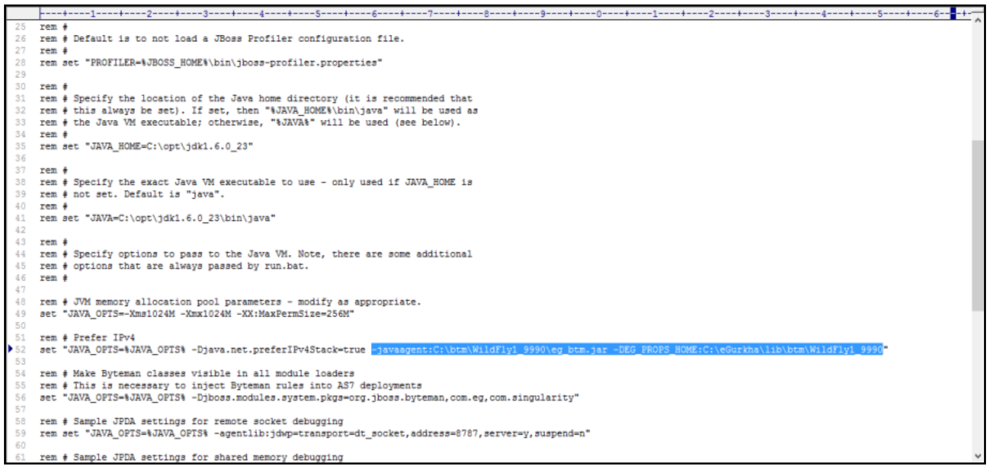
18. Then, in the file, enter the following lines, as depicted by Figure 2.42.

```
-javaagent:<<PATH TO THE eg_btm.jar FILE>>
-DEG_PROPS_HOME=<<PATH OF THE LOCAL FOLDER CONTAINING THE .PROPS FILES>>
```

For instance, if the eg_btm.jar file and the .props files are extracted into the **C:\btm\WildFly1_ 9990** directory, the above specification will be:

```
-javaagent:C\btm\WildFly1_9990\eg_btm.jar
```

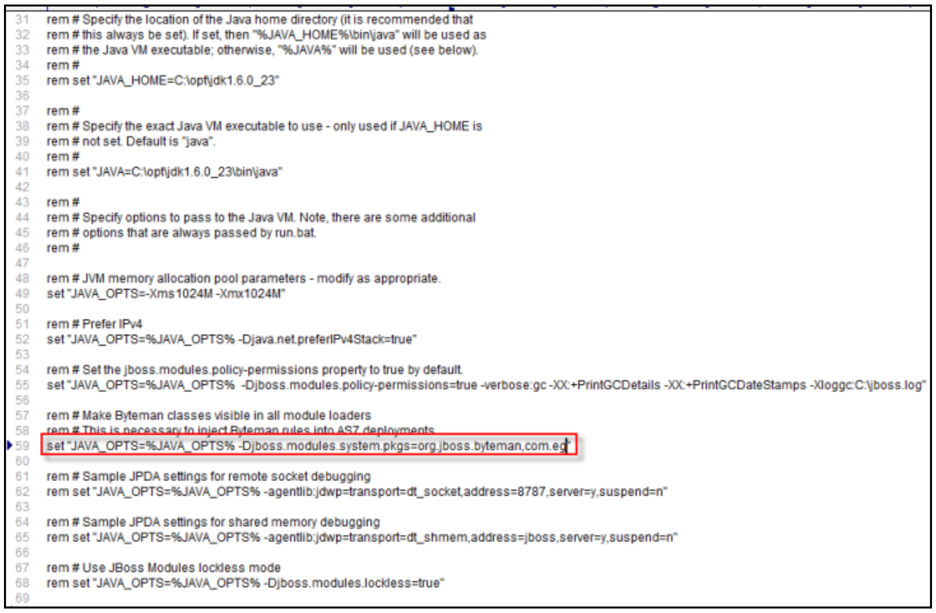
```
-DEG_PROPS_HOME=C:\btm\WildFly1_9990
```



```
rem #
rem # Default is to not load a JBoss Profiler configuration file.
rem #
rem set "PROFILER=%JBOSS_HOME%\bin\jboss-profiler.properties"

rem #
rem # Specify the location of the Java home directory (it is recommended that
rem # this always be set). If set, then "%JAVA_HOME%\bin\java" will be used as
rem # the Java VM executable; otherwise, "%JAVA%" will be used (see below).
rem #
rem set "JAVA_HOME=C:\opt\jdk1.6.0_23"

rem #
rem # Specify the exact Java VM executable to use - only used if JAVA_HOME is
rem # not set. Default is "java".
rem #
rem set "JAVA=C:\opt\jdk1.6.0_23\bin\java"

rem #
rem # Specify options to pass to the Java VM. Note, there are some additional
rem # options that are always passed by run.bat.
rem #

rem # JVM memory allocation pool parameters - modify as appropriate.
set "JAVA_OPTS=-Xms1024M -Xmx1024M -XX:MaxPermSize=256M"

rem # Prefer IPv4
set "JAVA_OPTS=%JAVA_OPTS% -Djava.net.preferIPv4Stack=true -javaagent:C:\btm\WildFly1_9990\eg_btm.jar -DEG_PROPS_HOME:C:\eGurkha\lib\btm\WildFly1_9990"

rem # Make Byteman classes visible in all module loaders
rem # This is necessary to inject Byteman rules into AS7 deployments
set "JAVA_OPTS=%JAVA_OPTS% -Djboss.modules.system.pkgs=org.jboss.byteman,com.eg,com.singularity"

rem # Sample JPDA settings for remote socket debugging
rem set "JAVA_OPTS=%JAVA_OPTS% -agentlib:jdwp=transport=dt_socket,address=8787,server=y,suspend=n"

rem # Sample JPDA settings for shared memory debugging
```

Figure 2.42: Editing the start-up script to BTM-enable a JBoss WildFly server that is monitored in an agent-based manner

19. Then, save the file.
20. Where multiple instances of WildFly are monitored, make sure you perform steps 15-19 above for each WildFly server instance.
21. Next, if the JBoss server is running in domain mode, open the **domain.conf** file in **JBoss_ Home\bin** directory. If the JBoss server is running in standalone mode, open the **standalone.conf** file in the **JBoss_Home\bin** directory.



```
rem # Specify the location of the Java home directory (it is recommended that
rem # this always be set). If set, then "%JAVA_HOME%\bin\java" will be used as
rem # the Java VM executable; otherwise, "%JAVA%" will be used (see below).
rem #
rem set "JAVA_HOME=C:\opt\jdk1.6.0_23"

rem #
rem # Specify the exact Java VM executable to use - only used if JAVA_HOME is
rem # not set. Default is "java".
rem #
rem set "JAVA=C:\opt\jdk1.6.0_23\bin\java"

rem #
rem # Specify options to pass to the Java VM. Note, there are some additional
rem # options that are always passed by run.bat.
rem #

rem # JVM memory allocation pool parameters - modify as appropriate.
set "JAVA_OPTS=-Xms1024M -Xmx1024M"

rem # Prefer IPv4
set "JAVA_OPTS=%JAVA_OPTS% -Djava.net.preferIPv4Stack=true"

rem # Set the jboss.modules.policy-permissions property to true by default.
set "JAVA_OPTS=%JAVA_OPTS%  -Djboss.modules.policy-permissions=true -verbose:gc -XX:+PrintGCDetails -XX:+PrintGCDateStamps -Xloggc:C:\jboss.log"

rem # Make Byteman classes visible in all module loaders
rem # This is necessary to inject Byteman rules into AS7 deployments
set "JAVA_OPTS=%JAVA_OPTS% -Djboss.modules.system.pkgs=org.jboss.byteman,com.eg"

rem # Sample JPDA settings for remote socket debugging
rem set "JAVA_OPTS=%JAVA_OPTS% -agentlib:jdwp=transport=dt_socket,address=8787,server=y,suspend=n"

rem # Sample JPDA settings for shared memory debugging
rem set "JAVA_OPTS=%JAVA_OPTS% -agentlib:jdwp=transport=dt_shmem,address=jboss,server=y,suspend=n"

rem # Use JBoss Modules lockless mode
rem set "JAVA_OPTS=%JAVA_OPTS% -Djboss.modules.lockless=true"
```

Figure 2.43: Editing the domain.conf file or standalone.conf file

22. Append **",com.eg"** to the following line in the file, as depicted by Figure 2.43:

```
set "JAVA_OPTS=%JAVA_OPTS% -Djboss.modules.system.pkgs="org.jboss.byteman"
```

23. Finally, save the file and restart the JBoss WildFly server.

2.7.2 BTM-Enabling a JBoss WildFly Server Running on a Unix Platform

If the target JBoss WildFly server is running on a Unix operating system, then follow the steps below to BTM-enable that server:

1. Login to any system in your environment that supports a browser and has network access to the eG manager.
2. Open a browser on that system, connect to the eG manager, and login to the eG admin interface.
3. Manage the target JBoss WildFly server as a separate component using the eG administrative interface. When managing, make a note of the **Nick name** and **Port number** that you provide.
4. If multiple JBoss WildFly server instances are operating on a single host, and you want to monitor each of those instances, then you will have to manage each instance as a separate JBoss WildFly server component using the eG administrative interface. When doing so, make a note of the **Nick name** and **Port number** using which you managed each instance.
5. Next, log out of the eG admin interface and the system.

6. Log into the target JBoss WildFly server. Then, create a **btm** directory anywhere on the target server - say, **/opt/btm**. Under this directory, create a sub-folder. Make sure that you name this sub- folder in the following format: *<Managed_ Component_ NickName>_ <Managed_ Component_Port>*. For instance, if you have managed the JBoss WildFly server using the nick name *WildFly1* and the port number *9990*, the new directory under the **btm** directory should be named as *WildFly1_9990*.
7. If you have managed multiple WildFly server instances running on a single host, then you will have to create multiple sub-directories under the **btm** directory - one each for every instance. Each of these sub-directories should be named after the **Nick name** and **Port number** using which the corresponding instance has been managed in eG.
8. Once the new sub-directory(ies) is created, log out of the JBoss WildFly server. Log back into the system you used in step 1 above. Open a browser on the system, connect to the eG manager, and login to the eG admin interface again.
9. Follow the Agents -> BTM Profiler Download menu sequence in the eG admin interface.
10. Figure 2.40 will appear listing the servers that can be instrumented for APM by eG. In this list, locate the JBoss WildFly server that you want to BTM-enable. Once you locate the server, click the **Download** icon corresponding to that server to download the **APM Profiler Agent** to that server. If multiple JBoss WildFly server instances on a single host are managed, then you will have to download the APM Profiler Agent separately for each of the managed instances.
11. Upon clicking the **Download** icon in Figure 2.40, a zip file named **javaagent_<Nick_name_of_ WildFly_server>_<Port_number_of_WildFly_server** will get downloaded. For instance, if you have managed the JBoss WildFly server using the nickname 'WildFly1' and the port number '9990', then the name of the zip file will be *javaagent_WildFly1_9990*. Where multiple JBoss WildFly server instances have been managed, you will be downloading multiple zip files - one each for every instance. The names of these zip files will automatically carry the nick name and port number you assigned to the corresponding JBoss WildFly server instance.
12. Next, using FTP tools like putty or WinSCP, transfer the downloaded zip file (s) to the corresponding sub-directory(ies) (of the **btm** directory), which you created on the JBoss WildFly server in steps 6 and 7 above. For example, the zip file named *javaagent_WildFly1_9990*, should be transferred to the**/opt/btm/WildFly1_9990** directory on the target JBoss WildFly server.
13. Log out of the system and log back into the JBoss WildFly server.
14. Extract the contents of each zip file into the same sub-directory to which that zip file was copied.
15. Figure 2.41 depicts the extracted contents of the zip file.

16. From Figure 2.41, it is evident that the zip file contains an **eg_btm.jar** file and a few property files, namely - **btmOther.props** , **btmLogging.props** , **config.props** , **custom.props** , **exclude.props**, and **threshold.props** files.

17. Next, proceed to BTM- enable the JBoss WildFly server/instance. For that, edit the **btmOther.props** file in the sub-directory (of the **btm** directory) that corresponds to that server/ instance. You will find the following lines in the file:

```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Below property is BTM Server Socket Port, through which eG Agent Communicates
# Restart is required, if any changes in this property
# Default port is "13931"
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#
BTM_Port=13931
#
```

By default, the **BTMPort** parameter is set to 13931. If you want to enable eG Java BTM on a different port, then specify the same here. In this case, when configuring the **Java Business Transactions** test or the **Key Java Business Transactions** test for that application server, make sure you configure the **BTMPORT** parameter of the test with this port number.

Note:

When BTM-enabling multiple instances on the same server, make sure you configure a different **BTM Port** for each instance.

Also, by default, the **Designated_Agent** parameter will be empty; do not disturb this default setting. In this case therefore, the eG Java BTM will treat the host from which the very first 'measure request' comes in as the **Designated_Agent**.

Note:

In case a specific **Designated_Agent** is not provided, and the eG Java BTM treats the host from which the very first 'measure request' comes in as the **Designated_Agent**, then if such a **Designated_Agent** is stopped or uninstalled for any reason, the eG Java BTM will wait for a maximum of 10 measure periods for that 'deemed' **Designated_Agent** to request for metrics. If no requests come in for 10 consecutive measure periods, then the eG Java BTM will begin responding to 'measure requests' coming in from any other eG agent.

18. Finally, save the **btmOther.props** file.

19. Then, you need to configure the JBoss WildFly server with the path to the **eg_btm.jar** and **.props** files. To achieve this, you need to edit the start-up script of the server. The first step towards that is to open the start-up script.

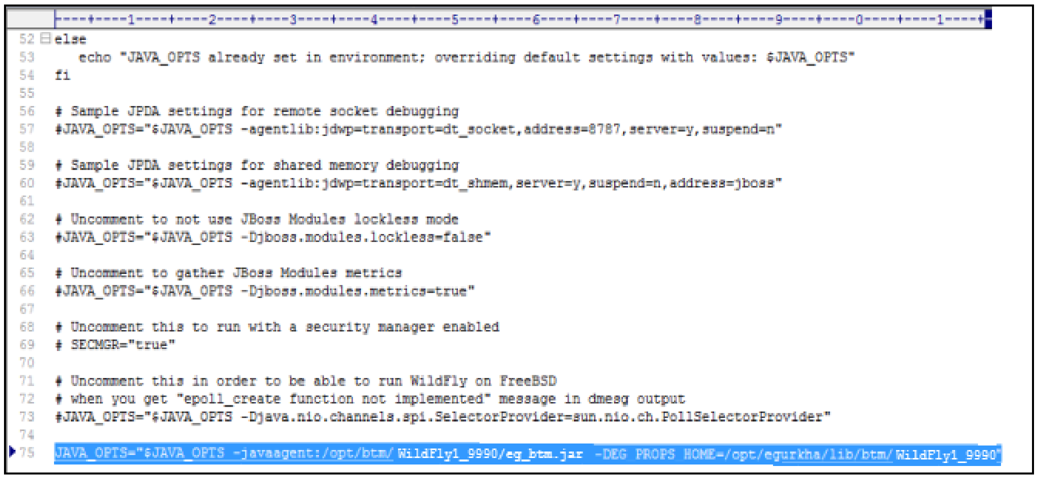
20. Then, in the file, enter the following lines, as depicted by Figure 2.44.

```
JAVA_OPTS="$JAVA_OPTS -javaagent:<<PATH TO the eg_btm.jar>> -DEG_PROPS_HOME=<<PATH TO
LOCAL FOLDER CONTAINING THE
```

```
.PROPS FILES>>"
```

For instance, if the eg_btm.jar file and .props files are extracted into the **/opt/btm/WildFly1_9990** directory, the above specification will be:

```
JAVA_OPTS="$JAVA_OPTS -javaagent:/opt/btm/WildFly1_9990/eg_btm.jar -DEG_PROPS_
HOME=/opt/btm/WildFly1_9990"
```



```
else
    echo "JAVA_OPTS already set in environment; overriding default settings with values: $JAVA_OPTS"
fi

# Sample JPDA settings for remote socket debugging
#JAVA_OPTS="$JAVA_OPTS -agentlib:jdwp=transport=dt_socket,address=8787,server=y,suspend=n"

# Sample JPDA settings for shared memory debugging
#JAVA_OPTS="$JAVA_OPTS -agentlib:jdwp=transport=dt_shmem,server=y,suspend=n,address=jboss"

# Uncomment to not use JBoss Modules lockless mode
#JAVA_OPTS="$JAVA_OPTS -Djboss.modules.lockless=false"

# Uncomment to gather JBoss Modules metrics
#JAVA_OPTS="$JAVA_OPTS -Djboss.modules.metrics=true"

# Uncomment this to run with a security manager enabled
# SECMGR="true"

# Uncomment this in order to be able to run WildFly on FreeBSD
# when you get "epoll_create function not implemented" message in dmesg output
#JAVA_OPTS="$JAVA_OPTS -Djava.nio.channels.spi.SelectorProvider=sun.nio.ch.PollSelectorProvider"

JAVA_OPTS="$JAVA_OPTS -javaagent:/opt/btm/ WildFly1_9990/eg_btm.jar  -DEG PROPS HOME=/opt/egurkha/lib/btm/ WildFly1_9990"
```

Figure 2.44: Editing the start-up script to BTM-enable a JBoss WildFly server on Unix that is monitored in an agent-based manner

21. In Unix environments, if the eG agent is deployed on the same host as the JBoss WildFly server, then both the agent and the server will be running using different user privileges. In this situation, by default, the eG Java BTM logs will not be created. In order to create the same, insert the following entry after the -DEG_PROPS_HOME specification, but before the closing quotes.

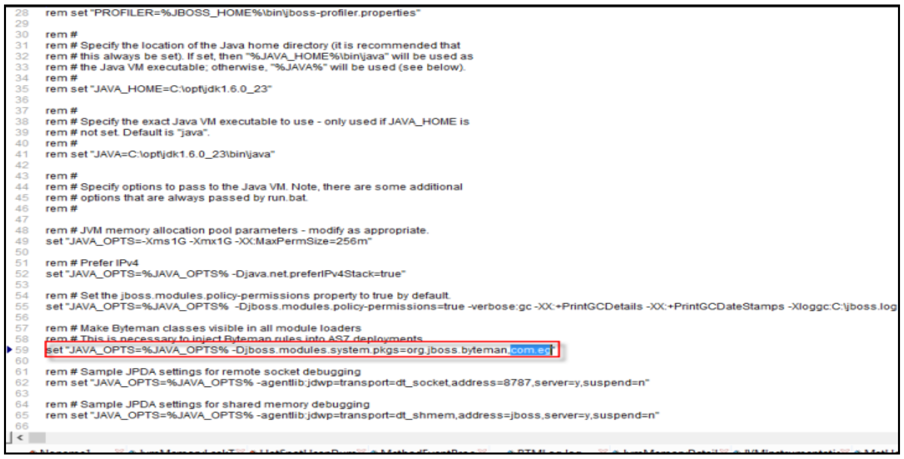
```
-DEG_LOG_HOME=<<LogFile_Path>>
```

Before providing this specification, make sure you create a folder for BTM logs - say, **eGBTMLogs** - in any directory to which the target application server has access. Then, against, -DEG_LOG_HOME, provide the full path to the **eGBTMLogs** directory. Where multiple instances on the same server are to be BTM-enabled, you can use the same directory for writing log files of all instances.

For example, to create log files in the **/App001/eGBTMLogs** directory, the complete specification will be as follows:

```
AVA_OPTS="$JAVA_OPTS -JAVA_OPTS="$JAVA_OPTS -javaagent:/opt/btm/WildFly1_9990/eg_
btm.jar -DEG_PROPS_HOME=/opt/btm/WildFly1_9990 -DEG_LOG_HOME=/App001/eGBTMLogs"
```

22. Then, save the file.
23. Where multiple instances of WildFly are monitored, make sure you perform steps 17-22 above for each WildFly server instance.
24. Next, if the JBoss server is running in domain mode, open the **domain.conf** file in **JBoss_Home/bin** directory. If the JBoss server is running in standalone mode, open the **standalone.conf** file in the **JBoss_Home/bin** directory.



```
rem set "PROFILER=%JBOSS_HOME%\bin\jboss-profiler.properties"

rem #
rem # Specify the location of the Java home directory (it is recommended that
rem # this always be set). If set, then "%JAVA_HOME%\bin\java" will be used as
rem # the Java VM executable; otherwise, "%JAVA%" will be used (see below).
rem #
rem set "JAVA_HOME=C:\opt\jdk1.6.0_23"

rem #
rem # Specify the exact Java VM executable to use - only used if JAVA_HOME is
rem # not set. Default is "java".
rem #
rem set "JAVA=C:\opt\jdk1.6.0_23\bin\java"

rem #
rem # Specify options to pass to the Java VM. Note, there are some additional
rem # options that are always passed by run.bat.
rem #

rem # JVM memory allocation pool parameters - modify as appropriate.
set "JAVA_OPTS=-Xms1G -Xmx1G -XX:MaxPermSize=256m"

rem # Prefer IPv4
set "JAVA_OPTS=%JAVA_OPTS% -Djava.net.preferIPv4Stack=true"

rem # Set the jboss.modules.policy-permissions property to true by default.
set "JAVA_OPTS=%JAVA_OPTS%  -Djboss.modules.policy-permissions=true -verbose:gc -XX:+PrintGCDetails -XX:+PrintGCDateStamps -Xloggc:C:\jboss.log

rem # Make Byteman classes visible in all module loaders
rem # This is necessary to inject Byteman rules into AS7 deployments
set "JAVA_OPTS=%JAVA_OPTS% -Djboss.modules.system.pkgs=org.jboss.byteman,com.eg"

rem # Sample JPDA settings for remote socket debugging
rem set "JAVA_OPTS=%JAVA_OPTS% -agentlib:jdwp=transport=dt_socket,address=8787,server=y,suspend=n"

rem # Sample JPDA settings for shared memory debugging
rem set "JAVA_OPTS=%JAVA_OPTS% -agentlib:jdwp=transport=dt_shmem,address=jboss,server=y,suspend=n"
```

Figure 2.45: Editing the domain.conf file or standalone.conf file

25. Append **",com.eg"** to the following line in the file, as depicted by Figure 2.43:

```
set "JAVA_OPTS=%JAVA_OPTS% -Djboss.modules.system.pkgs="org.jboss.byteman"
```

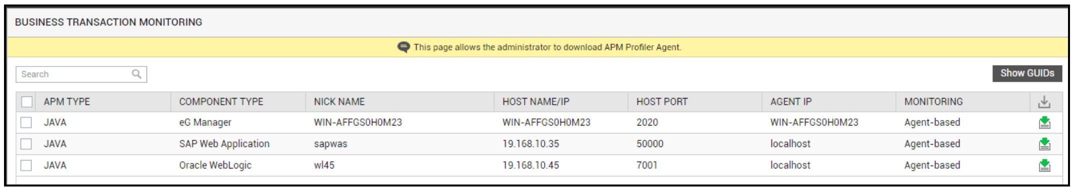
26. Finally, save the file and restart the JBoss WildFly server.

2.8 Installing eG BTM on a Multi- Server SAP Web Application Server Instance

In a multi-server environment, two/more Java server processes run within a single SAP WAS instance.. For the eG agent to monitor business transactions to a ,multi-server SAP WAS instance, you need to BTM-enable each server Java process in that instance, separately.

To achieve this, follow the steps detailed below.

1. First, login to the SAP WAS instance to be BTM-enabled.
2. Open a browser on that instance and connect to the eG management console.
3. Login to the eG admin interface.
4. Follow the Agents -> BTM Profiler Download menu sequence in the eG admin interface.
5. Figure 2.46 will appear listing the servers that can be instrumented for APM by eG. In this list, locate the SAP Web Application server instance that you want to BTM-enable. Once you locate the server instance, click the **Download** icon corresponding to that instance to download the **APM Profiler Agent** to that instance.

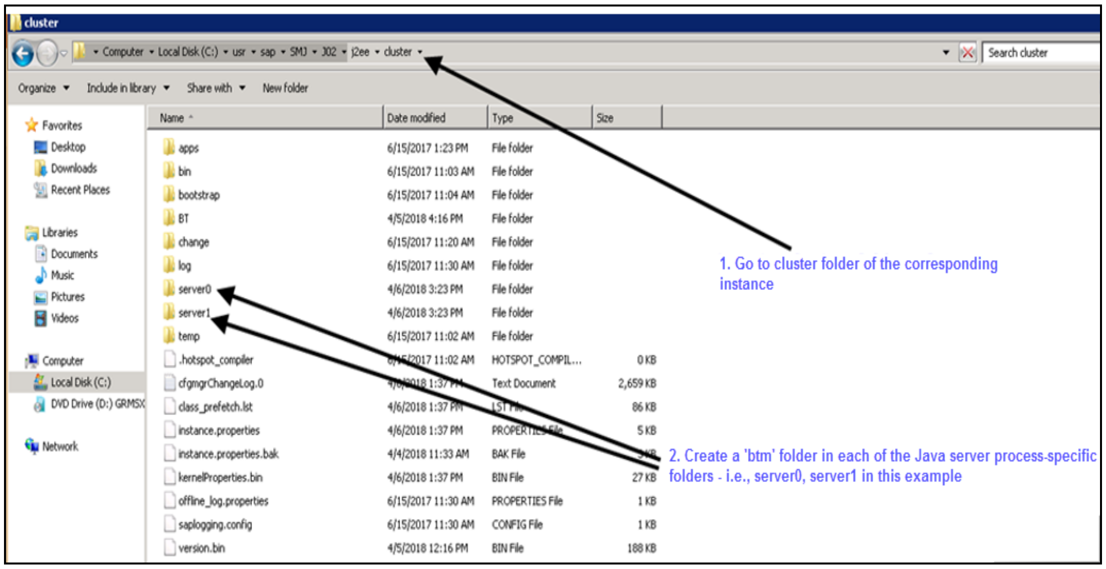


This page allows the administrator to download APM Profiler Agent.

APM TYPE	COMPONENT TYPE	NICK NAME	HOST NAME/IP	HOST PORT	AGENT IP	MONITORING	
JAVA	eG Manager	WIN-AFFGS0H0M23	WIN-AFFGS0H0M23	2020	WIN-AFFGS0H0M23	Agent-based	
JAVA	SAP Web Application	sapwas	19.168.10.35	50000	localhost	Agent-based	
JAVA	Oracle WebLogic	wl45	19.168.10.45	7001	localhost	Agent-based	

Figure 2.46: Download the APM Profiler Agent to the SAP WAS instance

6. A zip file named **javaagent_<Nick_name_of_SAPWAS_instance>_<Port_number_of_ SAPWAS_ instance** will get downloaded. For instance, if you have managed the SAP WAS instance using the nickname 'sapwas' and the port number '50000', then the name of the zip file will be *javaagent_sapwas_50000*. Extract the contents of this file to any location on the SAP WAS instance.
7. Next, navigate to the **cluster** folder of the SAP WAS instance. Within this folder, you will find Java server process-specific sub-folders. Create a folder named **btm** under each of these sub-folders.

8. 

Name	Date modified	Type	Size
apps	6/15/2017 1:23 PM	File folder	
bin	6/15/2017 11:03 AM	File folder	
bootstrap	6/15/2017 11:04 AM	File folder	
BT	4/5/2018 4:16 PM	File folder	
change	6/15/2017 11:20 AM	File folder	
log	6/15/2017 11:30 AM	File folder	
server0	4/6/2018 3:23 PM	File folder	
server1	4/6/2018 3:23 PM	File folder	
temp	6/15/2017 11:02 AM	File folder	
.hotspot_compiler	6/15/2017 11:02 AM	HOTSPOT_COMPIL...	0 KB
cfgmgrChangeLog.0	4/6/2018 1:37 PM	Text Document	2,659 KB
class_prefetch.lst	4/6/2018 1:37 PM	LST File	86 KB
instance.properties	4/6/2018 1:37 PM	PROPERTIES File	5 KB
instance.properties.bak	4/4/2018 11:33 AM	BAK File	5 KB
kernelProperties.bin	4/6/2018 1:37 PM	BIN File	27 KB
offline_log.properties	6/15/2017 11:30 AM	PROPERTIES File	1 KB
saplogging.config	6/15/2017 11:30 AM	CONFIG File	1 KB
version.bin	4/5/2018 12:16 PM	BIN File	188 KB

1. Go to cluster folder of the corresponding instance

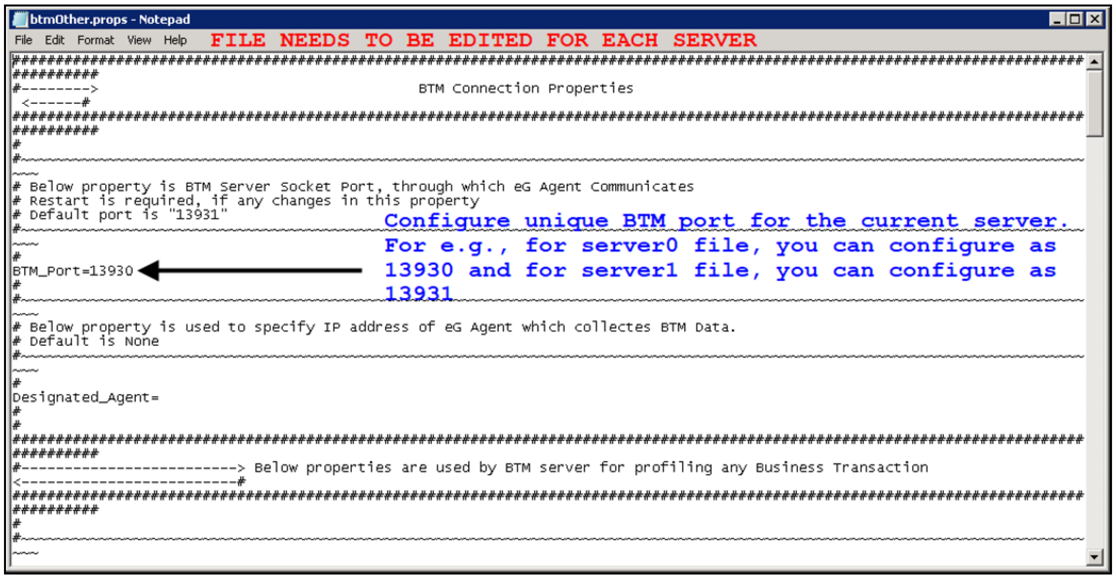
2. Create a 'btm' folder in each of the Java server process-specific folders - i.e., server0, server1 in this example

Figure 2.47: Navigating to the cluster folder in the <SAP_WAS_INSTANCE_INSTALL_DIR>

9. Copy the contents of the **javaagent_sapwas_50000** zip to the **btm** folder of each of the Java server process-specific folders (indicated by Figure 2.47).

10. Next, follow the steps below for each Java server process:

    ○ Edit the **btmOther.props** file in the btm folder of the Java server process.


btmOther.props - Notepad
File Edit Format View Help
FILE NEEDS TO BE EDITED FOR EACH SERVER
Configure unique BTM port for the current server.
For e.g., for server0 file, you can configure as 13930 and for server1 file, you can configure as 13931

```
##########################################################################################################
##########
#--------->                                          BTM Connection Properties
 <------#
##########################################################################################################
##########
#
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~
# Below property is BTM Server Socket Port, through which eG Agent Communicates
# Restart is required, if any changes in this property
# Default port is "13931"
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~
#
BTM_Port=13930
#
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~
# Below property is used to specify IP address of eG Agent which collectes BTM Data.
# Default is None
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~
#
Designated_Agent=
#
#
##########################################################################################################
##########
#-------------------------> Below properties are used by BTM server for profiling any Business Transaction
<--------------------------#
##########################################################################################################
##########
#
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~
```

Figure 2.48: Configuring the BTM port for a server process

- In the **btmOther.props** file, search for the **BTM_Port** parameter. Once found, specify a unique BTM port for the Java server process.

  Typically, each Java server process in a multi-server SAP WAS instance is assigned a node index - eg., the node index for server0 is 0, for server1 is 1, for server2 is 2, and so on. When configuring a BTM port for a Java server process, its recommended that the node index of that server process be set as the last digit of the BTM port number. For instance, the BTM port for server0 can be 13930, server1 can be 13931, server2 can be 13932 and so on.

- Finally, save the file.

- Repeat this procedure for every server process in the SAP WAS instance.

11. Next, connect to the SAP Netweaver administrator tool by providing the URL: http://<IP_ address_of_tool>:<Port_number_of_tool>/nwa/sysinfo

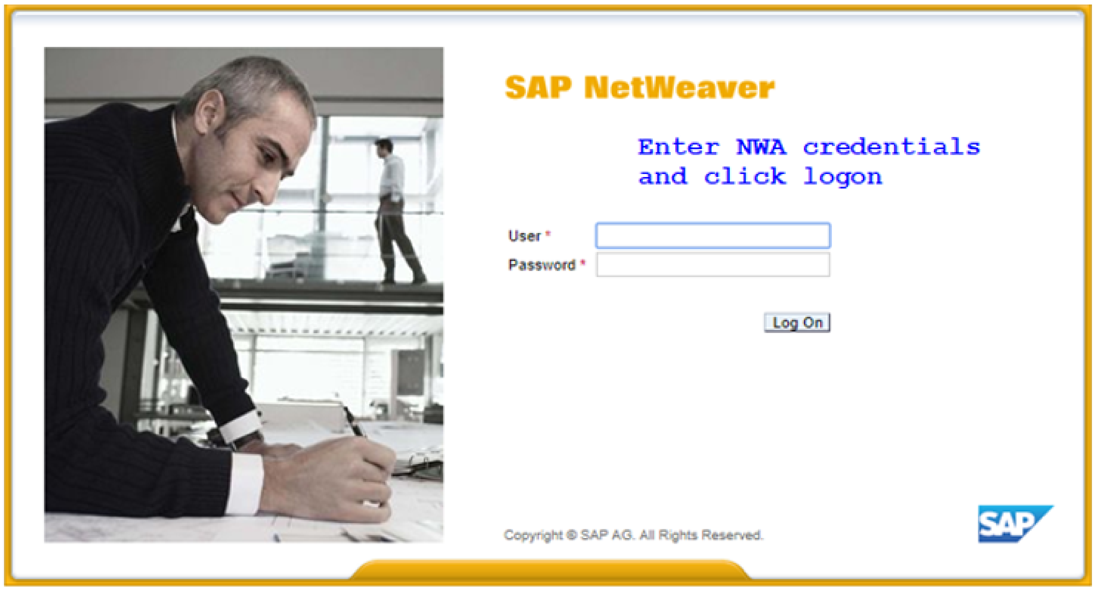
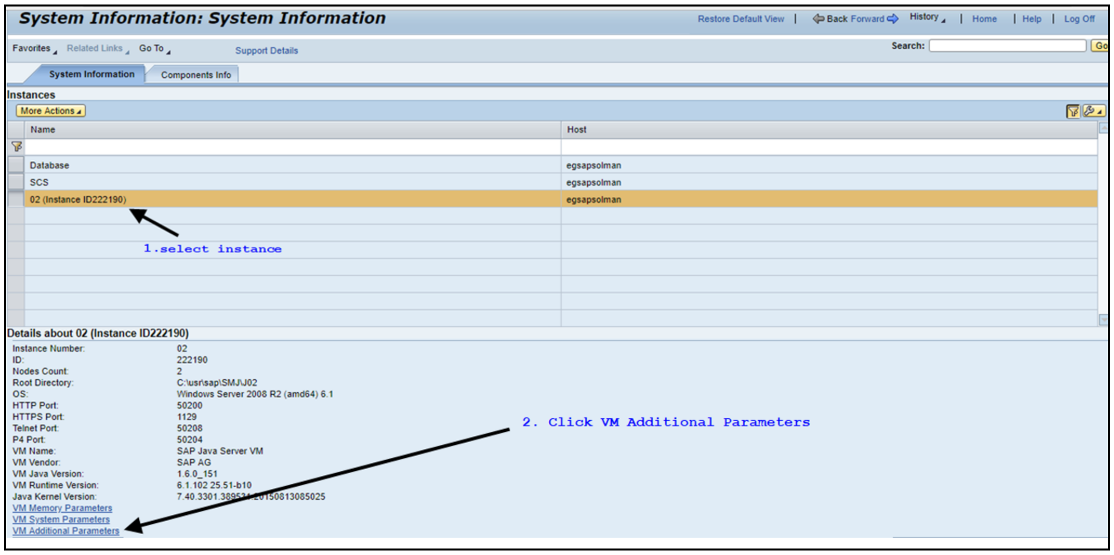

SAP NetWeaver
Enter NWA credentials and click logon
User *
Password *
Log On
Copyright © SAP AG. All Rights Reserved.
SAP

Figure 2.49: The login page of the SAP Netweaver administrator tool

12. A login screen then appears (see Figure 2.49). Login to the system information application using administrator or j2ee_admin credentials. Upon successful login, Figure 2.50 will appear.


System Information: System Information
Restore Default View | Back Forward History | Home | Help | Log Off
Favorites Related Links Go To Support Details Search: Go
System Information
Components Info
More Actions
1.select instance
2. Click VM Additional Parameters

Instances

Name	Host
Database	egsapsolman
SCS	egsapsolman
02 (Instance ID222190)	egsapsolman

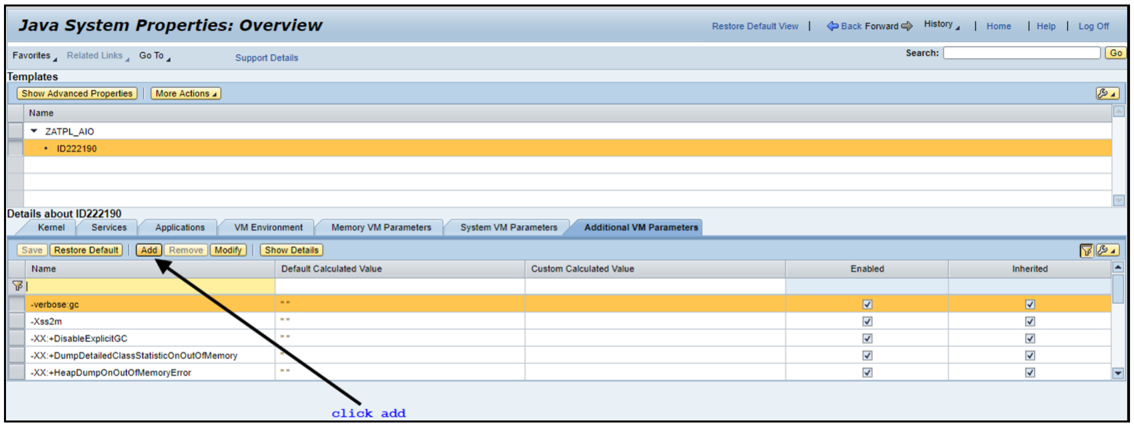
Details about 02 (Instance ID222190)

Instance Number:	02
ID:	222190
Nodes Count:	2
Root Directory:	C:\usr\sap\SMJ\J02
OS:	Windows Server 2008 R2 (amd64) 6.1
HTTP Port:	50200
HTTPS Port:	1129
Telnet Port:	50208
P4 Port:	50204
VM Name:	SAP Java Server VM
VM Vendor:	SAP AG
VM Java Version:	1.6.0_151
VM Runtime Version:	6.1.102 25.51-b10
Java Kernel Version:	7.40.3301.389534.20150813085025

VM Memory Parameters
VM System Parameters
VM Additional Parameters

Figure 2.50: Clicking the VM Additional Parameters link

13. Select the instance to be BTM-enabled from the list of **Instances** in Figure 2.50. Then, click the **VM Additional Parameters** link indicated by Figure 2.50. This will open the **Additional VM Parameters** tab page (see Figure 2.51). Click the **Add** button in that tab page to add a new VM parameter.


Java System Properties: Overview
Restore Default View | Back Forward History | Home | Help | Log Off
Favorites Related Links Go To Support Details Search: Go
Show Advanced Properties
More Actions
Kernel
Services
Applications
VM Environment
Memory VM Parameters
System VM Parameters
Additional VM Parameters
Save Restore Default Add Remove Modify Show Details
click add

Templates

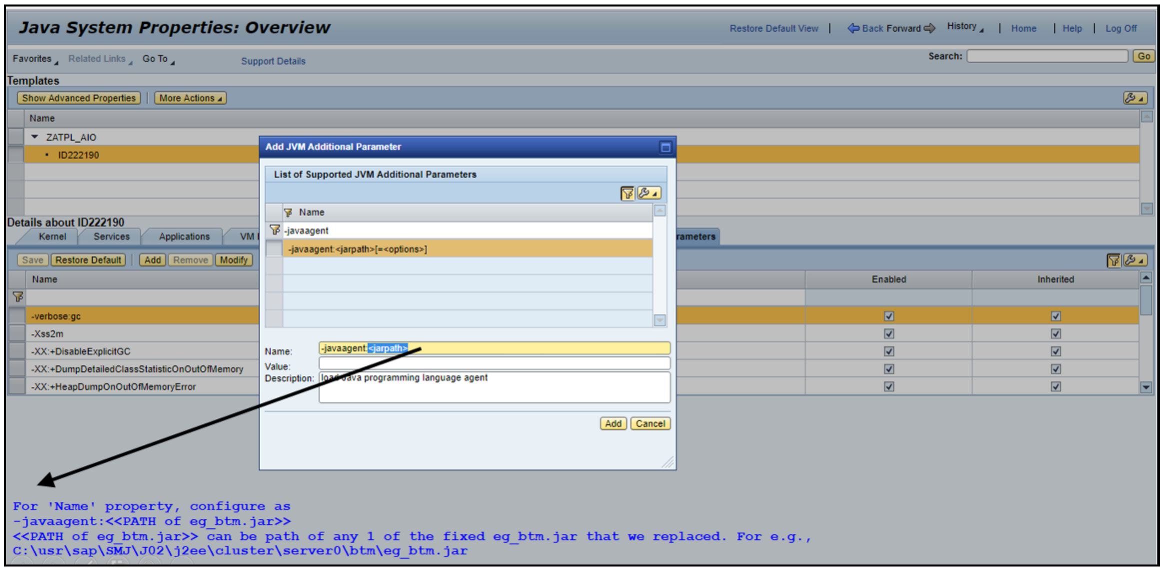
Name
ZATPL_AIO
ID222190

Details about ID222190

Name	Default Calculated Value	Custom Calculated Value	Enabled	Inherited
-verbose:gc	" "		✓	✓
-Xss2m	" "		✓	✓
-XX:+DisableExplicitGC	" "		✓	✓
-XX:+DumpDetailedClassStatisticOnOutOfMemory	"		✓	✓
-XX:+HeapDumpOnOutOfMemoryError	" "		✓	✓

Figure 2.51: Clicking the Add button in the Additional VM Parameters tab page

14. Figure 2.52 will then appear.


Java System Properties: Overview
Restore Default View | Back Forward | History | Home | Help | Log Off
Favorites Related Links Go To Support Details
Search: Go
Templates
Show Advanced Properties | More Actions
Details about ID222190
Kernel Services Applications VM
Save Restore Default | Add Remove Modify
Add JVM Additional Parameter
List of Supported JVM Additional Parameters
Add Cancel

Name
ZATPL_AIO
ID222190

Name	Enabled	Inherited
-verbose:gc	✓	✓
-Xss2m	✓	✓
-XX:+DisableExplicitGC	✓	✓
-XX:+DumpDetailedClassStatisticOnOutOfMemory	✓	✓
-XX:+HeapDumpOnOutOfMemoryError	✓	✓

Name
-javaagent
-javaagent:<jarpath>[=<options>]

Name: -javaagent:<jarpath>
Value:
Description: load Java programming language agent

```
For 'Name' property, configure as
-javaagent:<<PATH of eg_btm.jar>>
<<PATH of eg_btm.jar>> can be path of any 1 of the fixed eg_btm.jar that we replaced. For e.g.,
C:\usr\sap\SMJ\J02\j2ee\cluster\server0\btm\eg_btm.jar
```

Figure 2.52: Adding a new VM parameter

15. Specify the following in the **Name** text box in Figure 2.52:

    -javaagent:<Path_to_eg_btm.jar_file>

    Here, you can specify the full path to the **eg_btm.jar file** that is in the **btm** folder of any of the Java server processes in the target instance.

16. Then, in the **Value** text box, specify the following:

    EG_PROPS_HOME=<Path_to_the_btmOther.props_file>,EG_LOG_HOME=<Path_to_the_btmLogging.props_file>

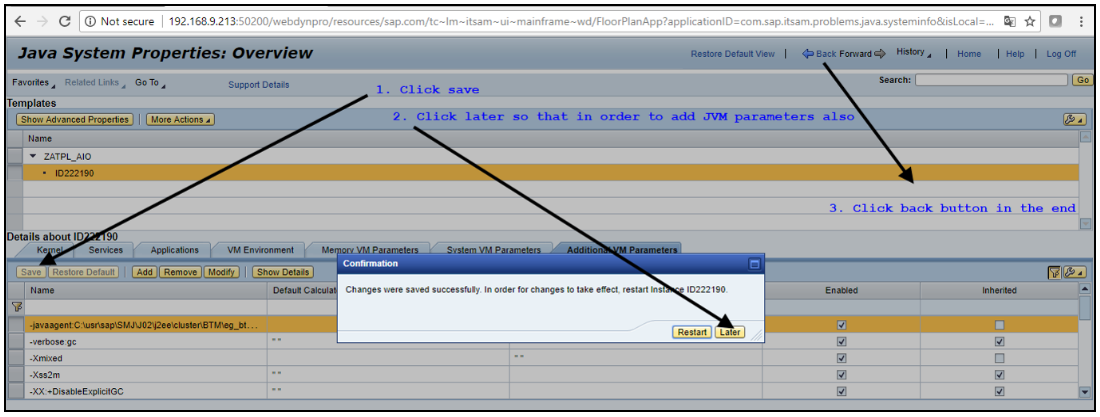
    The path to these files will be different for each Java server process in the instance. Hence, when referring to the server process-specific sub-folder that contains these files, your path specification should include the ${NODE_INDEX} variable. This variable represents the node index of a Java server process - eg., the node index of server process server0 is 0, for server1 it is 1, and so on. SAP WAS will automatically substitute this variable with the node index of the corresponding server process, at runtime. A sample specification is provided below:

    EG_PROPS_HOME=C:\usr\sap\SMJ\J02\j2ee\cluster\server${NODE_INDEX}\btm,EG_LOG_HOME=C:\usr\sap\SMJ\J02\j2ee\cluster\server${NODE_INDEX}\btm

17. Finally, click the **Add** button in Figure 2.52 to add the new VM parameter.

18. When Figure 2.52 appears, click the **Save** button to save the changes. A **Confirmation**

message box (see Figure 2.53) will appear prompting you to confirm whether you want to restart the server now or later. Click **Later** in the message box and then click the **Back** button indicated by Figure 2.53.



1. Click save

2. Click later so that in order to add JVM parameters also

3. Click back button in the end

Changes were saved successfully. In order for changes to take effect, restart Instance ID222190.

Name	Default Calculat		Enabled	Inherited
-javaagent:C:\usr\sap\SMJ\J02\j2ee\cluster\BTM\eg_bt...			✓	☐
-verbose:gc	" "		✓	✓
-Xmixed		" "	✓	☐
-Xss2m	" "		✓	✓
-XX:+DisableExplicitGC	" "		✓	✓

Figure 2.53: Saving the changes

19. Restart the Java server processes that have been BTM-enabled.

Chapter 3: Monitoring Java Business Transactions

Once the eG Application Server Agent is installed and configured on the JVM nodes, it will start tracking transaction requests and storing transaction path and metrics in memory.

To collect these metrics, you then need to configure the eG agent to run the **Java Business Transactions** test. To focus on only those transactions you deem important, you can optionally configure the eG agent to run the **Key Java Business Transactions** test.

The metrics reported are then captured into an Application Transactions layer. This layer will appear as the first layer of the monitoring model of the application server that is BTM-enabled.

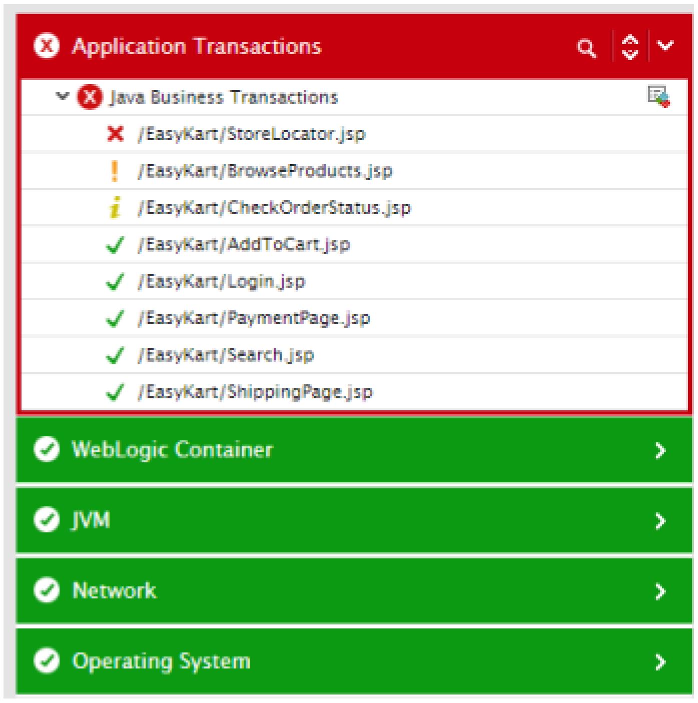

Application Transactions
Java Business Transactions
/EasyKart/StoreLocator.jsp
/EasyKart/BrowseProducts.jsp
/EasyKart/CheckOrderStatus.jsp
/EasyKart/AddToCart.jsp
/EasyKart/Login.jsp
/EasyKart/PaymentPage.jsp
/EasyKart/Search.jsp
/EasyKart/ShippingPage.jsp
WebLogic Container
JVM
Network
Operating System

Figure 3.1: The test mapped to the Application Transactions layer

This chapter discusses how to configure these tests and details the metrics reported by each test.

3.1 Java Business Transactions Test

The responsiveness of a transaction is the key determinant of user experience with that transaction; if response time increases, user experience deteriorates. To make users happy, a Java business transaction should be rapidly processed by each of the JVM nodes in its path. Processing bottlenecks on a single JVM node can slowdown/stall an entire business transaction or can cause serious transaction errors. This in turn can badly scar the experience of users. To avoid this, administrators should promptly identify slow/stalled/errored transactions, isolate the JVM node on which the slowness/error occurred, and uncover what caused the aberration on that node – is it owing to SQL queries executed by the node? Or is it because of external calls – eg., async calls, SAP JCO calls, HTTP calls, etc. - made by that node? The **Java Business Transactions** test helps with this!

This test runs on a BTM-enabled JVM in an IT infrastructure, tracks all the transaction requests received by that JVM, and groups requests based on user-configured pattern specifications. For each transaction pattern, the test then computes and reports the average time taken by that JVM node to respond to the transaction requests of that pattern. In the process, the test identifies the slow/stalled transactions of that pattern, and reports the count of such transactions and their responsiveness. Detailed diagnostics provided by the test accurately pinpoint the exact transaction URLs that are slow/stalled, the total round-trip time of each transaction, and also indicate when such transaction requests were received by that node. The slowest transaction in the group can thus be identified.

Moreover, to enable administrators to figure out if the slowness can be attributed to a bottleneck in SQL query processing, the test also reports the average time the transactions of each pattern took to execute SQL queries. If a majority of the queries are slow, then the test will instantly capture the same and notify administrators.

Additionally, the test promptly alerts administrators to error transactions of each pattern. To know which are the error transactions, the detailed diagnosis capability of the test can be used.

This way, the test effortlessly measures the performance of each transaction to a JVM node, highlights transactions that are under-performing, and takes administrators close to the root-cause of poor transaction performance.

Target of the Test : A BTM-enabled JVM

Agent deploying the test : An internal/remote agent

Output of the test : One set of results for each grouped URL

Test parameters:

Configurable parameters for the test

Parameter	Description
Test Period	How often should the test be executed.
Host	The host for which this test is to be configured.
BTM Port	Specify the port number specified as BTM_Port in the **btmOther.props** file on the JVM

Parameter	Description
	node being monitored. If the JVM is being monitored in an agent-based manner, then the *btmOther.props* file will be in the <EG_AGENT_INSTALL_DIR>\lib\bm directory.
Max URL Segments	This test groups transaction URLs based on the URL segments count configured for monitoring and reports aggregated response time metrics for every group. Using this parameter, you can specify the number of URL segments based on which the transactions are to be grouped. URL segments are the parts of a URL (after the base URL) or path delimited by slashes. So if you had the URL: *http://www.eazykart.com/web/shopping/sportsgear/login.jsp*, then http://www.eazykart.com will be the base URL or domain, /web will be the first URL segment, /shopping will be the second URL segment, and /sportsgear will be the third URL segment, and /login.jsp will be the fourth URL segment. By default, this parameter is set to 3. This default setting, when applied to the sample URL provided above, implies that the eG agent will aggregate response time metrics to all transaction URLs under /web/shopping/sportsgear. Note that the base URL or domain will not be considered when counting URL segments. This in turn means that, if the JVM node receives transaction requests for the URLs such as *http://www.eazykart.com/web/shopping/sportsgear/login.jsp, http://www.eazykart.com/web/shopping/sportsgear/jerseys.jsp, http://www.eazykart.com/web/shopping/sportsgear/shoes.jsp, http://www.eazykart.com/web/shopping/sportsgear/gloves.jsp*, etc., then the eG agent will track the requests and responses for all these URLs, aggregate the results, and present the aggregated metrics for the descriptor /web/shopping/sportsgear. This way, the test will create different transaction groups based on each of the third-level URL segments – eg. /web/shopping/weddings, /web/shopping/holiday, /web/shopping/gifts etc. – and will report aggregated metrics for each group so created. If you want, you can override the default setting by providing a different URL segment number here. For instance, your specification can be just 2. In this case, for the URL *http://www.eazykart.com/web/shopping/login.jsp*, the test will report metrics for the descriptor web/shopping.
Excluded Patterns	By default, this test does not track requests to the following URL patterns: **.ttf, *.otf, *.woff, *.woff2, *.eot, *.cff, *.afm, *.lwfn, *.ffil, *.fon, *.pfm, *.pfb, *.std, *.pro, *.xsf, *.jpg, *.jpeg, *.jpe, *.jif, *.jfif, *.jfi, *.jp2, *.j2k, *.jpf, *.jpx, *.jpm, *.jxr, *.hdp, *.wdp, *.mj2, *.webp, *.gif, *.png, *.apng, *.mng, *.tiff, *.tif, *.xbm, *.bmp, *.dib, *.svg, *.svgz, *.mpg, *.mpeg, *.mpeg2, *.avi, *.wmv, *.mov, *.rm, *.ram, *.swf, *.flv, *.ogg, *.webm, *.mp4, *.ts, *.mid, *.midi, *.rm, *.ram, *.wma, *.aac, *.wav, *.ogg, *.mp3, *.mp4, *.css, *.js, *.ico, *.cur, /egurkha** If required, you can remove one/more patterns from this default list, so that such

Parameter	Description
	patterns are monitored, or can append more patterns to this list in order to exclude them from monitoring.
Method Exec Cutoff (MS)	From the detailed diagnosis of slow/stalled/error transactions, you can drill down and perform deep execution analysis of a particular transaction. In this drill-down, the methods invoked by that slow/stalled/error transaction are listed in the order in which the transaction calls the methods. By configuring a Method Exec Cutoff, you can make sure that methods that have been executing for a duration greater the specified cutoff are alone listed when performing execution analysis. For instance, if you specify *5* here, then the **Execution Analysis** window for a slow/stalled/error transaction will list only those methods that have been executing for over 5 milliseconds. This way, you get to focus on only those methods that could have caused the slowness, without being distracted by inconsequential methods. By default, the value of this parameter is set to 250 ms.
SQL Execution Cutoff (MS)	Typically, from the detailed diagnosis of a slow/stalled/error transaction on a JVM node, you can drill down to view the SQL queries (if any) executed by that transaction from that node and the execution time of each query. By configuring a SQL Execution Cutoff (MS), you can make sure that queries that have been executing for a duration greater the specified cutoff are alone listed when performing query analysis. For instance, if you specify *5* here, then for a slow/stalled/error transaction, the **SQL Queries** window will display only those queries that have been executing for over 5 milliseconds. This way, you get to focus on only those queries that could have contributed to the slowness. By default, the value of this parameter is set to 10 ms.
Healthy URL Trace	By default, this flag is set to **No**. This means that eG will not collect detailed diagnostics for those transactions that are healthy. If you want to enable the detailed diagnosis capability for healthy transactions as well, then set this flag to **Yes**.
Max Healthy URLs per Test Period	**This parameter is applicable only if the Healthy URL Trace flag is set to 'Yes'**. Here, specify the number of top-n transactions that should be listed in the detailed diagnosis of the *Healthy transactions* measure, every time the test runs. By default, this is set to *50*, indicating that the detailed diagnosis of the *Healthy transactions* measure will by default list the top-50 transactions, arranged in the descending order of their response times.
Max Slow URLs per Test Period	Specify the number of top-n transactions that should be listed in the detailed diagnosis of the *Slow transactions* measure, every time the test runs. By default, this is set to *10*, indicating that the detailed diagnosis of the *Slow transactions* measure will by default list the top-10 transactions, arranged in the descending order of their response times.
Max Stalled URLs	Specify the number of top-n transactions that should be listed in the detailed diagnosis

Parameter	Description
per Test Period	of the *Stalled transactions* measure, every time the test runs. By default, this is set to *10*, indicating that the detailed diagnosis of the *Stalled transactions* measure will by default list the top-10 transactions, arranged in the descending order of their response times.
Max Error URLs per Test Period	Specify the number of top-n transactions that should be listed in the detailed diagnosis of the *Error transactions* measure, every time the test runs. By default, this is set to *10*, indicating that the detailed diagnosis of the *Error transactions* measure will by default list the top-10 transactions, in terms of the number of errors they encountered.
Show HTTP Status	If you want the detailed diagnosis of this test to report the HTTP response code that was returned when a transaction URL was hit, then set this flag to **Yes**. This will enable you to instantly identify HTTP errors that may have occurred when accessing a transaction URL. By default, this flag is set to **No**, indicating that the HTTP status code is not reported by default as part of detailed diagnostics.
Show Cookies	An HTTP cookie is a small piece of data sent from a website and stored on the user's computer by the user's web browser while the user is browsing. Most commonly, cookies are used to provide a way for users to record items they want to purchase as they navigate throughout a website (a virtual "shopping cart" or "shopping basket"). To keep track of which user is assigned to which shopping cart, the server sends a cookie to the client that contains a unique session identifier (typically, a long string of random letters and numbers). Because cookies are sent to the server with every request the client makes, that session identifier will be sent back to the server every time the user visits a new page on the website, which lets the server know which shopping cart to display to the user. Another popular use of cookies is for logging into websites. When the user visits a website's login page, the web server typically sends the client a cookie containing a unique session identifier. When the user successfully logs in, the server remembers that that particular session identifier has been authenticated, and grants the user access to its services. If you want to view and analyze the useful information that is stored in such HTTP response cookies that a web server sends, then set this flag to **Yes**. By default, this flag is set to **No**, indicating that cookie information is not reported by default as part of detailed diagnostics.
Show Headers	HTTP headers allow the client and the server to pass additional information with the request or the response. A request header is a header that contains more information about the resource to be fetched or about the client itself. If you want the additional information stored in a request header to be displayed as part of detailed diagnostics, then set this flag to **Yes**. By default, this flag is set to **No** indicating that request headers are not displayed by default in the detailed diagnosis.
Enable Thread CPU Monitoring	If this flag is set to **Yes**, then this test will additionally report the average time for which

Parameter	Description
	the transactions of a pattern were utilizing the CPU resources. This will point you to transaction patterns that are CPU-intensive, and will thus help you right-size your JVMs. By default however, this test will not report the average CPU time of transaction patterns. This is because, by default, the Enable Thread CPU Monitoring flag is set to **No** for this test.
Enable Thread Contention Monitoring	If this flag is set to **Yes**, then this test will additionally report the following: • The average time for which the transactions of a pattern were waiting, before they resumed execution; • The average time for which the transactions of a pattern were blocked from execution by another transaction; If transactions of a pattern are found to be much slower than the rest or are stalling, then the aforesaid metrics will help administrators determine what could have caused the slowness - is it because the transactions were waiting for too long? or is it because they were being blocked for too long? By default however, this test will not report the metrics described above, because the Enable Thread Contention Monitoring flag is set to **No** by default.
Advanced Settings	To optimize transaction performance and conserve space in the eG database, many restraints have been applied by default on the agent's ability to collect and report detailed diagnostics. Depending upon how well-tuned your eG database is and the level of visibility you require into transaction performance, you may choose to either retain these default settings or override them. If you choose not to disturb the defaults, then set the Advanced Settings flag to **No**. If you want to modify the defaults, then set the Advanced Settings flag to **Yes**.
POJO Method Tracing Limit and POJO Method Tracing Cutoff Time	**These parameters will appear only if the Advanced Settings flag is set to 'Yes'**. Typically, if the **Monitoring Mode** of this test is set to **Profiler**, then, as part of the detailed diagnostics of a transaction, eG reports the execution time of every POJO, non-POJO, and recursive (i.e. methods that call themselves) method call that a JVM node makes when processing that transaction. Of these, POJO method calls are the most expensive, as they are usually large in number. To ensure that attempts made to collect detailed measures related to POJO method calls do not impact the overall responsiveness of the monitored transaction, eG, by default, collects and reports the execution time of only the following POJO method calls: • The first 1000 POJO method calls made by the target JVM node for that transaction; (OR)

Parameter	Description
	• The POJO method calls that were made by the target JVM node within 10 seconds from the start of the monitored transaction on that node; Accordingly, the POJO Method Tracing Limit is set to 1000 by default, and the POJO Method Tracing Cutoff Time is set to 10 (seconds) by default. Of these two limits, whichever limit is reached first will automatically be applied by eG for determining when to stop POJO tracing. In other words, once a JVM node starts processing a transaction, the agent begins tracking the POJO method calls made by that node for that transaction. In the process, if the agent finds that the configured tracing limit is reached before the tracing cutoff time is reached, then the agent will stop tracking the POJO method calls, as soon as the tracing limit is reached. On the other hand, if the tracing limit is not reached, then the agent will continue tracking the POJO method calls until the tracing cutoff time is reached. At the end of the cutoff time, the agent will stop tracking the POJO method calls. For instance, if the JVM node makes 1000 POJO method calls within say, 6 seconds from when it began processing the transaction, then the eG agent will not wait for the cutoff time of 10 seconds to be reached; instead, it will stop tracing at the end of the thousandth POJO method call, and report the execution time of each of the 1000 calls alone. On the other hand, if the JVM node does not make over 1000 POJO method calls till the 10 second cutoff expires, then the eG agent continues tracking the POJO method calls till the end of 10 seconds, and reports the details of all those that were calls made till the cutoff time. Depending upon how many POJO calls you want to trace and how much overhead you want to impose on the agent and on the transaction, you can increase / decrease the POJO Method Tracing Limit and POJO Method Tracing Cutoff time specifications.
Non-POJO Method Tracing Limit	**This parameter will appear only if the Advanced Settings flag is set to 'Yes'**. By default, when reporting the detailed diagnosis of a transaction on a particular JVM node, this test reports the execution time of only the first 1000 non-POJO method calls (which includes JMS, JCO, HTTP, Java, SQL, etc.) that the target JVM node makes for that transaction. This is why, the Non-POJO Method Tracing Limit parameter is set to *1000* by default. If you want, you can change the tracing limit to enable the test to report the details of more or fewer non-POJO method calls made by a JVM node. While a high value for this parameter may take you closer to identifying the non-POJO method that could have caused the transaction to slowdown on a particular JVM node, it may also marginally increase the overheads of the transaction and the eG agent.
Recursive Method Tracing Limit	**This parameter will appear only if the Advanced Settings flag is set to 'Yes'**. A recursive method is a method that calls itself. By default, when reporting the detailed diagnosis of a transaction on a particular JVM node, this test reports the execution time of only the first 1000 recursive method calls (which includes JMS, JCO, HTTP, Java,

Parameter	Description
	SQL, etc.) that the target JVM node makes for that transaction. This is why, the Recursive Method Tracing Limit parameter is set to 1000 by default. If you want, you can change the tracing limit to enable the test to report the details of more or fewer recursive method calls made by a JVM node. While a high value for this parameter may take you closer to identifying the recursive method that could have caused the transaction to slowdown on a particular JVM node, it may also marginally increase the overheads of the transaction and the eG agent.
Exception Stacktrace Lines	**This parameter will appear only if the Advanced Settings flag is set to 'Yes'**. As part of detailed diagnostics, this test, by default, lists the first 10 stacktrace lines of each JavaScript error/exception that it captures on the target JVM node for a specific transaction, so as to enable easy and efficient troubleshooting. This is why, the Exception Stacktrace Lines parameter is set to *10* by default. If required, you can have this test display more or fewer stacktrace lines by overriding this default setting.
Included Exceptions	**This parameter will appear only if the Advanced Settings flag is set to 'Yes'**. By default, this test flags the transactions in which the following errors/exceptions are captured, as *Error transactions*: • All unhandled exceptions; • Both handled and unhandled SQL exceptions/errors This implies that if a programmatically-handled non-SQL exception occurs in a transaction, such a transaction, by default, will not be counted as an *Error transaction* by this test. Sometimes however, administrators may want to be alerted even if some non-SQL exceptions that have already been handled programmatically, occur. This can be achieved by configuring a comma-separated list of these exceptions in the Included Exceptions text box. Here, each exception you want to include has to be defined using its fully qualified exception class name. For instance, your Included Exceptions specification can be as follows: java.lang.NullPointerException, java.lang.IndexOutOfBoundsException. **Note that wild card characters cannot be used as part of your specification**. Once the exceptions to be included are configured, then this test will count all transactions in which such exceptions are captured as *Error transactions*.
Ignored Exceptions	**This parameter will appear only if the Advanced Settings flag is set to 'Yes'**. By default, this test flags the transactions in which the following errors/exceptions are captured, as *Error transactions*: • All unhandled exceptions;

Parameter	Description
	• Both handled and unhandled SQL exceptions/errors Sometimes however, administrators may want eG to disregard certain unhandled exceptions (or handled SQL exceptions), as they may not pose any threat to the stability of the transaction or to the web site/web application. To achieve this, administrators can configure a comma-separated list of such inconsequential exceptions in the Ignored Exceptions text box. Here, you need to configure each exception you want to exclude using its fully qualified exception class name. For instance, your Excluded Exceptions specification can be as follows: java.sql.SQLException,java.io.FileNotFoundException. **Note that wild card characters cannot be used as part of your specification**. Once the exceptions to be excluded are configured, then this test will exclude all transactions in which such exceptions are captured from its count of *Error transactions*.
Ignored Characters	**This parameter will appear only if the Advanced Settings flag is set to 'Yes'**. By default, eG excludes all transaction URLs that contain the '\' character from monitoring. If you want eG to ignore transaction URLs with any other special characters, then specify these characters as a comma-separated list in the Ignored Characters text box. For instance, your specification can be: \\,&,~
Max Grouped URLs per Measure Period	This parameter will appear only if the Advanced Settings flag is set to 'Yes'. This test groups URLs according to the Max URL Segments specification. These grouped URLs will be the descriptors of the test. For each grouped URL, response time metrics will be aggregated across all transaction URLs in that group and reported. When monitoring web sites/web applications to which the transaction volume is normally high, this test may report metrics for hundreds of descriptors. If all these descriptors are listed in the **Layers** tab page of the eG monitoring console, it will certainly clutter the display. To avoid this, by default, the test displays metrics for a maximum of 50 descriptors – i.e., 50 grouped URLs alone – in the eG monitoring console, during every measure period. This is why, the Max Grouped URLs per Measure Period parameter is set to 50 by default. To determine which 50 grouped URLs should be displayed in the eG monitoring console, the eG BTM follows the below-mentioned logic: • Top priority is reserved for URL groups with error transactions. This means that eG BTM first scans URL groups for error transactions. If error transactions are found in 50 URL groups, then eG BTM computes the aggregated response time of each of the 50 groups, sorts the error groups in the descending order of their response time, and displays all these 50 groups alone as the descriptors of this test, in the sorted

Parameter	Description
	order. • On the other hand, if error transactions are found in only one / a few URL groups – say, only 20 URL groups – then, eG BTM will first arrange these 20 grouped URLs in the descending order of their response time. It will then compute the aggregated response time of the transactions in each of the other groups (i.e., the error-free groups) that were auto-discovered during the same measure period. These other groups are then arranged in the descending order of the aggregated response time of their transactions. Once this is done, eG BTM will then pick the top-30 grouped URLs from this sorted list. In this case, when displaying the descriptors of this test in the **Layers** tab page, the 20 error groups are first displayed (in the descending order of their response time), followed by the 30 'error-free' groups (also in the descending order of their response time). At any given point in time, you can increase/decrease the maximum number of descriptors this test should supportby modifying the value of the Max Grouped URLs per Measure Period parameter.
Max SQl Queries per Transaction	**This parameter will appear only if the Advanced Settings flag is set to 'Yes'**. Typically, from the detailed diagnosis of a slow/stalled/error transaction on a JVM node, you can drill down to view the SQL queries (if any) executed by that transaction from that node and the execution time of each query. By default, eG picks the first *500* SQL queries executed by the transaction, compares the execution time of each query with the SQL Execution Cutoff configured for this test, and displays only those queries with an execution time that is higher than the configured cutoff. This is why, the Max SQL Queries per Transaction parameter is set to *500* by default. To improve agent performance, you may want the SQL Execution Cutoff to be compared with the execution time of a less number of queries – say, 200 queries. Similary, to increase the probability of capturing more number of long-running queries, you may want the sql execution cutoff to be compared with the execution time of a large number of queries – say, 1000 queries. For this, you just need to modify the Max SQL Queries per Transaction specification to suit your purpose.
Timeout	By default, the eG agent will wait for 1000 milliseconds for a response from the eG Application Server agent. If no response is received, then the test will timeout. You can change this timeout value, if required.
DD Frequency	Refers to the frequency with which detailed diagnosis measures are to be generated for this test. The default is *1:1*. This indicates that, by default, detailed measures will be generated every time this test runs, and also every time the test detects a problem. You can modify this frequency, if you so desire. Also, if you intend to disable the

Parameter	Description
	detailed diagnosis capability for this test, you can do so by specifying *none* against DD frequency.
Detailed Diagnosis	To make diagnosis more efficient and accurate, the eG Enterprise suite embeds an optional detailed diagnostic capability. With this capability, the eG agents can be configured to run detailed, more elaborate tests as and when specific problems are detected. To enable the detailed diagnosis capability of this test for a particular server, choose the **On** option. To disable the capability, click on the **Off** option. The option to selectively enable/disable the detailed diagnosis capability will be available only if the following conditions are fulfilled: • The eG manager license should allow the detailed diagnosis capability • Both the normal and abnormal frequencies configured for the detailed diagnosis measures should not be 0.

Measures reported by the test

Measurement	Description	Measurement Unit	Interpretation
All transactions	Indicates the total number of requests received for transactions of this pattern during the last measurement period.	Number	By comparing the value of this measure across transaction patterns, you can identify the most popular transaction patterns. Using the detailed diagnosis of this measure, you can then figure out which specific transactions of that pattern are most requested. For the **Summary** descriptor, this measure will reveal the total number of transaction requests received by the target JVM during the last measurement period. This is a good indicator of the transaction workload on that JVM.
Avg response time	Indicates the average time taken by the transactions	Msecs	Compare the value of this measure across patterns to isolate the type of

Measurement	Description	Measurement Unit	Interpretation
	of this pattern to complete execution.		transactions that were taking too long to execute. You can then use the detailed diagnosis of the All transactions measure of that group to know how much time each transaction in that group took to execute. This will lead you to the slowest transaction. For the **Summary** descriptor, this measure will reveal the average responsiveness of all the transaction requests received by the target JVM during the last measurement period. An abnormally low value for this measure for the Summary descriptor could indicate a serious processing bottleneck on the target JVM.
Healthy transactions	Indicates the number of healthy transactions of this pattern.	Number	By default, this measure will report the count of transactions with a response time less than 4000 milliseconds. You can change this default setting by modifying the thresholds of the *Avg response time* measure using the eG admin interface. For the **Summary** descriptor, this measure will report the total number of healthy transactions on the target JVM.
Healthy transactions percentage	Indicates what percentage of the total number of transactions of this pattern is healthy.	Percent	To know which are the healthy transactions, use the detailed diagnosis of this measure. For the **Summary** descriptor, this measure will report the overall percentage of healthy transactions on the target JVM.
Slow transactions	Indicates the number of transactions of this pattern that were slow during the last measurement period.	Number	By default, this measure will report the number of transactions with a response time higher than 4000 milliseconds and lesser than 60000

Measurement	Description	Measurement Unit	Interpretation
			milliseconds. You can change this default setting by modifying the thresholds of the *Avg response time* measure using the eG admin interface. A high value for this measure is a cause for concern, as too many slow transactions means that user experience with the web application is poor. For the **Summary** descriptor, this measure will report the total number of slow transactons on the target JVM. This is a good indicator of the processing power of the target JVM.
Slow transactions response time - avg	Indicates the average time taken by the slow transactions of this pattern to execute.	Msecs	For the **Summary** descriptor, this measure will report the average response time of all the slow transactions on the target JVM.
Slow transactions percentage	Indicates what percentage of the total number of transactions of this pattern is currently slow.	Percent	Use the detailed diagnosis of this measure to know which precise transactions of a pattern are slow. You can drill down from a slow transaction to know what is causing the slowness. For the **Summary** descriptor, this measure will report the overall percentage of slow transactions on the monitored JVM.
Error transactions	Indicates the number of transactions of this pattern that experienced errors during the last measurement period.	Number	A high value is a cause for concern, as too many error transactions to a web application can significantly damage the user experience with that application. For the **Summary** descriptor, this measure will report the total number of error transactons on the target JVM. This is a good indicator of how error-prone the target JVM is.
Error transactions response time - avg	Indicates the average duration for which the	Msecs	The value of this measure will help you discern if error transactions were also

Measurement	Description	Measurement Unit	Interpretation
	transactions of this pattern were processed before an error condition was detected.		slow. For the **Summary** descriptor, this measure will report the average response time of all error transactions on the target JVM.
Error transactions percentage	Indicates what percentage of the total number of transactions of this pattern is experiencing errors.	Percent	Use the detailed diagnosis of this measure to isolate the error transactions. You can even drill down from an error transaction in the detailed diagnosis to determine the cause of the error. For the **Summary** descriptor, this measure will report the overall percentage of transactions of this pattern on the target JVM that is currently experiencing errors.
Stalled transactions	Indicates the number of transactions of this pattern that were stalled during the last measurement period.	Number	By default, this measure will report the number of transactions with a response time higher than 60000 milliseconds. You can change this default setting by modifying the thresholds of the *Avg response time* measure using the eG admin interface. A high value is a cause for concern, as too many stalled transactions means that user experience with the web application is poor. For the **Summary** descriptor, this measure will report the total number of stalled transactons on the target JVM.
Stalled transactions response time - avg:	Indicates the average time taken by the stalled transactions of this pattern to execute.	Msecs	For the **Summary** descriptor, this measure will report the average response time of all stalled transactions on the target JVM.
Stalled transactions percentage	Indicates what percentage of the total number of transactions of this pattern is stalling.	Percent	Use the detailed diagnosis of this measure to know which precise transactions of a pattern are stalled. You can drill down from a stalled transaction to know what is causing

Measurement	Description	Measurement Unit	Interpretation
			that transaction to stall. For the **Summary** descriptor, this measure will report the overall percentage of transactions of this pattern on the target JVM that is stalling.
Slow SQL statements executed	Indicates the number of slow SQL queries that were executed by the transactions of this pattern during the last measurement period.	Number	For the **Summary** descriptor, this measure will report the total number of slow SQL queries executed by all transactions to the target JVM.
Avg slow SQL statement time	Indicates the average execution time of the slow SQL queries that were run by the transactions of this pattern.	Msecs	If there are too many slow transactions of a pattern, you may want to check the value of this measure for that pattern to figure out if query execution is slowing down the transactions. Use the detailed diagnosis of the *Slow transactions* measure to identify the precise slow transaction. Then, drill down from that slow transaction to confirm whether/not database queries have contributed to the slowness. Deep-diving into the queries will reveal the slowest queries and their impact on the execution time of the transaction.
Avg CPU time	Indicates the average time for which transactions of this pattern were utilizing the CPU.	Msecs	Compare the value of this measure across transaction patterns to accurately identify the CPU-intensive transaction patterns. For the **Summary** descriptor, this measure will report the average time for which all the transactions on the target JVM used the CPU. **Note:** This measure is reported only under the following circumstances:

Measurement	Description	Measurement Unit	Interpretation
			• The Enable Thread CPU Monitoring flag of this test is set to **Yes**; • The target application server's JVM implementation supports CPU time monitoring of threads; to verify JVM support for CPU time monitoring, use the procedure described in the Verifying JVM Support for CPU Time and Thread Contention Monitoring.
Avg block time	Indicates the average duration for which transactions of this pattern were blocked and could not execute.	Msecs	If the *Avg response time* for any transaction pattern is very high, you may want to check the value of this measure for that pattern. This will help you figure out whether/not prolonged blocking is causing transactions of that pattern to slow down or stall. For the **Summary** descriptor, this measure will report the average time for which all the transactions on the target JVM were blocked. **Note:** This measure is reported only under the following circumstances: • The Enable Thread Contention Monitoring flag of this test is set to **Yes**; • The target application server's JVM implementation supports thread contention monitoring; to verify JVM support for thread contention monitoring, use the procedure

Measurement	Description	Measurement Unit	Interpretation
			described in the Verifying JVM Support for CPU Time and Thread Contention Monitoring topic.
Avg wait time	Indicates the average duration for which transactions of this pattern were waiting before they resumed execution.	Msecs	If the *Avg response time* for any transaction pattern is very high, you may want to check the value of this measure for that pattern. This will help you figure out whether/not a very high waiting time is what is causing the transactions to slow down/stall. For the **Summary** descriptor, this measure will report the average time for which all the transactions on the target JVM were waiting. **Note:** This measure is reported only under the following circumstances: • The Enable Thread Contention Monitoring flag of this test is set to **Yes**; • The target application server's JVM implementation supports thread contention monitoring; to verify JVM support for thread contention monitoring, use the procedure described in the Verifying JVM Support for CPU Time and Thread Contention Monitoring topic.
Total transactions per minute	Indicates the number of transactions of this pattern that are executed per minute.	Number	For the **Summary** descriptor, this measure will report the total number of transactions that were executed per minute. This is a good indicator of the transaction processing ability of the

Measurement	Description	Measurement Unit	Interpretation
			target application server.
Error transactions per minute	Indicates the number of error transactions of this pattern that are executed per minute.	Number	A very low value is desired for this measure. Compare the value of this measure across transaction patterns to find that pattern of transactions that is experiencing errors frequently. For the **Summary** descriptor, this measure will report the total number of error transactions that were executed per minute.

3.2 Java Key Business Transactions Test

For any business-critical application, some transactions will always be considered key from the point of view of user experience and business impact. For instance, in the case of a retail banking web application, fund transfers executed online are critical transactions that have to be tracked closely for delays / errors, as problems in the transaction will cost both consumers and the company dearly. Using the **Java Key Business Transactions** test, administrators can perform focused monitoring of such critical transactions alone.

For each transaction URL pattern configured for monitoring on a JVM node, this test reports the count of requests for that transaction pattern, and the count and percentage of transactions of that pattern that were slow / stalling / error-prone. Detailed diagnostics provided by the test highlight the slow / stalled / error transactions of a pattern, and pinpoint the precise reason why that key transaction slowed down / stalled / encountered errors - is it because of an inefficient database query? is it because of a processing bottleneck on the JVM node? or is it owing to slow remote service calls? This way, the test enables you to quickly detect inconsistencies in the performance of your critical business transactions and accurately isolate its root-cause, so that you can fix the issues well before users notice them.

Target of the Test : A BTM-enabled JVM

Agent deploying the test : An internal/remote agent

Output of the test : One set of results for each URL pattern configured for monitoring.

Test parameters:

Configurable parameters for the test

Parameter	Description
Test Period	How often should the test be executed.
Host	The host for which this test is to be configured.
BTM Port	Specify the port number specified as BTM_Port in the **btmOther.props** file on the JVM node being monitored. If the JVM is being monitored in an agent-based manner, then the *btmOther.props* file will be in the <EG_AGENT_INSTALL_DIR>\lib\bm directory.
URL Patterns	Provide a comma-separated list of *PatternName:URLPattern* pairs to be monitored. The *PatternName* can be any name that uniquely identifies the pattern. These *PatternNames* will be the descriptors of this test. For the URLPattern, you can either provide the exact URL to be monitored , or can provide a pattern. For instance, if you want to monitor requests to distinct and specific web pages - say, login.jsp and payment.jsp of a web application - then you can specify the exact URL of these web pages as your *URL PATTERNS*. In this case your specification will be,*Login:/web/login.jsp,Payment:/web/payment.jsp*. On the other hand, if you want to monitor requests to all payment-related web pages in a web application - say, *payment.jsp, creditcardpayment.jsp, debitcardpayment.jsp, onlinepayment.jsp*, and more - and you want the metrics to be grouped under a single head called *Payment*, then you can specify a pattern instead of the exact URL. In this case, your *URL PATTERNS* specification will be *Payment:*payment**. The leading '*' in the specification signifies any number of leading characters, while the trailing '*' signifies any number of trailing characters. This means that the specification in our example will track requests to all pages with names that contain the word *payment*. Your *URLPattern* can also be **expr* or *expr** or **expr1*expr2** or *expr1*expr2*, etc.
Key Excluded Patterns	By default, this test does not track requests to the following URL patterns: **.ttf, *.otf, *.woff, *.woff2, *.eot, *.cff, *.afm, *.lwfn, *.ffil, *.fon, *.pfm, *.pfb, *.std, *.pro, *.xsf, *.jpg, *.jpeg, *.jpe, *.jif, *.jfif, *.jfi, *.jp2, *.j2k, *.jpf, *.jpx, *.jpm, *.jxr, *.hdp, *.wdp, *.mj2, *.webp, *.gif, *.png, *.apng, *.mng, *.tiff, *.tif, *.xbm, *.bmp, *.dib, *.svg, *.svgz, *.mpg, *.mpeg, *.mpeg2, *.avi, *.wmv, *.mov, *.rm, *.ram, *.swf, *.flv, *.ogg, *.webm, *.mp4, *.ts, *.mid, *.midi, *.rm, *.ram, *.wma, *.aac, *.wav, *.ogg, *.mp3, *.mp4, *.css, *.js, *.ico, *.cur, /egurkha** 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.
Method Exec Cutoff (MS)	From the detailed diagnosis of slow/stalled/error transactions, you can drill down and perform deep execution analysis of a particular transaction. In this drill-down, the methods invoked by that slow/stalled/error transaction are listed in the order in which

Parameter	Description
	the transaction calls the methods. By configuring a Method Exec Cutoff (MS), you can make sure that methods that have been executing for a duration greater the specified cutoff are alone listed when performing execution analysis. For instance, if you specify *5* here, then the **Execution Analysis** window for a slow/stalled/error transaction will list only those methods that have been executing for over 5 milliseconds. This way, you get to focus on only those methods that could have caused the slowness, without being distracted by inconsequential methods. By default, the value of this parameter is set to 250 ms.
SQL Execution Cutoff (MS)	Typically, from the detailed diagnosis of a slow/stalled/error transaction on a JVM node, you can drill down to view the SQL queries (if any) executed by that transaction from that node and the execution time of each query. By configuring a SQL Execution Cutoff (MS), you can make sure that queries that have been executing for a duration greater the specified cutoff are alone listed when performing query analysis. For instance, if you specify *5* here, then for a slow/stalled/error transaction, the **SQL Queries** window will display only those queries that have been executing for over 5 milliseconds. This way, you get to focus on only those queries that could have contributed to the slowness. By default, the value of this parameter is set to 10 ms.
Healthy URL Trace	By default, this flag is set to **No**. This means that eG will not collect detailed diagnostics for those transactions that are healthy. If you want to enable the detailed diagnosis capability for healthy transactions as well, then set this flag to **Yes**.
Max Healthy URLs per Test Period	**This parameter is applicable only if the Healthy URL Trace flag is set to 'Yes'**. Here, specify the number of top-n transactions that should be listed in the detailed diagnosis of the Healthy transactions measure, every time the test runs. By default, this is set to 50, indicating that the detailed diagnosis of the *Healthy transactions* measure will by default list the top-50 transactions, arranged in the descending order of their response times.
Max Slow URLs per Test Period	Specify the number of top-n transactions that should be listed in the detailed diagnosis of the *Slow transactions* measure, every time the test runs. By default, this is set to *10*, indicating that the detailed diagnosis of the *Slow transactions* measure will by default list the top-10 transactions, arranged in the descending order of their response times.
Max Stalled URLs per Test Period	Specify the number of top-n transactions that should be listed in the detailed diagnosis of the *Stalled transactions* measure, every time the test runs. By default, this is set to *10*, indicating that the detailed diagnosis of the *Stalled transactions* measure will by default list the top-10 transactions, arranged in the descending order of their response times.
Max Error URLs per	Specify the number of top-n transactions that should be listed in the detailed

Parameter	Description
Test Period	diagnosis of the *Error transactions* measure, every time the test runs. By default, this is set to *10*, indicating that the detailed diagnosis of the *Error transactions* measure will by default list the top-10 transactions, in terms of the number of errors they encountered.
Show HTTP Status	If you want the detailed diagnosis of this test to report the HTTP response code that was returned when a transaction URL was hit, then set this flag to **Yes**. This will enable you to instantly identify HTTP errors that may have occurred when accessing a transaction URL. By default, this flag is set to **No**, indicating that the HTTP status code is not reported by default as part of detailed diagnostics.
Show Cookies	An HTTP cookie is a small piece of data sent from a website and stored on the user's computer by the user's web browser while the user is browsing. Most commonly, cookies are used to provide a way for users to record items they want to purchase as they navigate throughout a website (a virtual "shopping cart" or "shopping basket"). To keep track of which user is assigned to which shopping cart, the server sends a cookie to the client that contains a unique session identifier (typically, a long string of random letters and numbers). Because cookies are sent to the server with every request the client makes, that session identifier will be sent back to the server every time the user visits a new page on the website, which lets the server know which shopping cart to display to the user. Another popular use of cookies is for logging into websites. When the user visits a website's login page, the web server typically sends the client a cookie containing a unique session identifier. When the user successfully logs in, the server remembers that that particular session identifier has been authenticated, and grants the user access to its services. If you want to view and analyze the useful information that is stored in such HTTP response cookies that a web server sends, then set this flag to **Yes**. By default, this flag is set to **No**, indicating that cookie information is not reported by default as part of detailed diagnostics.
Show Headers	HTTP headers allow the client and the server to pass additional information with the request or the response. A request header is a header that contains more information about the resource to be fetched or about the client itself. If you want the additional information stored in a request header to be displayed as part of detailed diagnostics, then set this flag to **Yes**. By default, this flag is set to **No** indicating that request headers are not displayed by default in the detailed diagnosis.
Enable Thread CPU Monitoring	If this flag is set to **Yes**, then this test will additionally report the average time for which the transactions of a pattern were utilizing the CPU resources. This will point you to transaction patterns that are CPU-intensive, and will thus help you right-size your JVMs. By default however, this test will not report the average CPU time of transaction patterns. This is because, by default, the Enable Thread CPU Monitoring flag is set to **No** for this test.

Parameter	Description
Enable Thread Contention Monitoring	If this flag is set to **Yes**, then this test will additionally report the following: • The average time for which the transactions of a pattern were waiting, before they resumed execution; • The average time for which the transactions of a pattern were blocked from execution by another transaction; If transactions of a pattern are found to be much slower than the rest or are stalling, then the aforesaid metrics will help administrators determine what could have caused the slowness - is it because the transactions were waiting for too long? or is it because they were being blocked for too long? By default however, this test will not report the metrics described above, because the Enable Thread Contention Monitoring flag is set to **No** by default.
Advanced Settings	To optimize transaction performance and conserve space in the eG database, many restraints have been applied by default on the agent's ability to collect and report detailed diagnostics. Depending upon how well-tuned your eG database is and the level of visibility you require into transaction performance, you may choose to either retain these default settings or override them. If you choose not to disturb the defaults, then set the Advanced Settings flag to **No**. If you want to modify the defaults, then set this flag to **Yes**.
POJO Method Tracing Limit and POJO Method Tracing Cutoff Time	**These parameters will appear only if the Advanced Settings flag is set to 'Yes'**. Typically, if the monitoring mode of this test is set to Profiler , then, as part of the detailed diagnostics of a transaction, eG reports the execution time of every POJO, non-POJO, and recursive (i.e. methods that call themselves) method call that a JVM node makes when processing that transaction. Of these, POJO method calls are the most expensive, as they are usually large in number. To ensure that attempts made to collect detailed measures related to POJO method calls do not impact the overall responsiveness of the monitored transaction, eG, by default, collects and reports the execution time of only the following POJO method calls: • The first 1000 POJO method calls made by the target JVM node for that transaction; (OR) • The POJO method calls that were made by the target JVM node within 10 seconds from the start of the monitored transaction on that node; Accordingly, the POJO Method Tracing Limit is set to 1000 by default, and the POJO Method Tracing Cutoff Time is set to 10 (seconds) by default. Of these two limits,

Parameter	Description
	whichever limit is reached first will automatically be applied by eG for determining when to stop POJO tracing. In other words, once a JVM node starts processing a transaction, the agent begins tracking the POJO method calls made by that node for that transaction. In the process, if the agent finds that the configured tracing limit is reached before the tracing cutoff time is reached, then the agent will stop tracking the POJO method calls, as soon as the tracing limit is reached. On the other hand, if the tracing limit is not reached, then the agent will continue tracking the POJO method calls until the tracing cutoff time is reached. At the end of the cutoff time, the agent will stop tracking the POJO method calls. For instance, if the JVM node makes 1000 POJO method calls within say, 6 seconds from when it began processing the transaction, then the eG agent will not wait for the cutoff time of 10 seconds to be reached; instead, it will stop tracing at the end of the thousandth POJO method call, and report the execution time of each of the 1000 calls alone. On the other hand, if the JVM node does not make over 1000 POJO method calls till the 10 second cutoff expires, then the eG agent continues tracking the POJO method calls till the end of 10 seconds, and reports the details of all those that were calls made till the cutoff time. Depending upon how many POJO calls you want to trace and how much overhead you want to impose on the agent and on the transaction, you can increase / decrease the POJO Method Tracing Limit and POJO Method Tracing Cutoff Time specifications.
Non-POJO Method Tracing Limit	**This parameter will appear only if the Advanced Settings flag is set to 'Yes'**. By default, when reporting the detailed diagnosis of a transaction on a particular JVM node, this test reports the execution time of only the first 1000 non-POJO method calls (which includes JMS, JCO, HTTP, Java, SQL, etc.) that the target JVM node makes for that transaction. This is why, the non-pojo method tracing limit parameter is set to 1000 by default. If you want, you can change the tracing limit to enable the test to report the details of more or fewer non-POJO method calls made by a JVM node. While a high value for this parameter may take you closer to identifying the non-POJO method that could have caused the transaction to slowdown on a particular JVM node, it may also marginally increase the overheads of the transaction and the eG agent.
Recursive Method Tracing Limit	**This parameter will appear only if the Advanced Settings flag is set to 'Yes'**. A recursive method is a method that calls itself. By default, when reporting the detailed diagnosis of a transaction on a particular JVM node, this test reports the execution time of only the first 1000 recursive method calls (which includes JMS, JCO, HTTP, Java, SQL, etc.) that the target JVM node makes for that transaction. This is why, the Recursive Method Tracing Limit parameter is set to 1000 by default. If you want, you can change the tracing limit to enable the test to report the details of more or fewer recursive method calls made by a JVM node. While a high value for this parameter may take you closer to identifying the recursive method that could have caused the transaction to slowdown on a particular JVM node, it may also marginally increase the

Parameter	Description
	overheads of the transaction and the eG agent.
Exception Stacktrace Lines	**This parameter will appear only if the Advanced Settings flag is set to 'Yes'**. As part of detailed diagnostics, this test, by default, lists the first 10 stacktrace lines of each JavaScript error/exception that it captures on the target JVM node for a specific transaction, so as to enable easy and efficient troubleshooting. This is why, the Exception Stacktrace Lines parameter is set to *10* by default. If required, you can have this test display more or fewer stacktrace lines by overriding this default setting.
Included Exceptions	**This parameter will appear only if the Advanced Settings flag is set to 'Yes'**. By default, this test flags the transactions in which the following errors/exceptions are captured, as *Error transactions*: • All unhandled exceptions; • Both handled and unhandled SQL exceptions/errors This implies that if a programmatically-handled non-SQL exception occurs in a transaction, such a transaction, by default, will not be counted as an *Error transaction* by this test. Sometimes however, administrators may want to be alerted even if some non-SQL exceptions that have already been handled programmatically, occur. This can be achieved by configuring a comma-separated list of these exceptions in the Included Exceptions text box. Here, each exception you want to include has to be defined using its fully qualified exception class name. For instance, your Included Exceptions specification can be as follows: java.lang.NullPointerException, java.lang.IndexOutOfBoundsException. **Note that wild card characters cannot be used as part of your specification**. Once the exceptions to be included are configured, then this test will count all transactions in which such exceptions are captured as *Error transactions*.
Ignored Exceptions	**This parameter will appear only if the Advanced settings flag is set to 'Yes'**. By default, this test flags the transactions in which the following errors/exceptions are captured, as *Error transactions*: • All unhandled exceptions; • Both handled and unhandled SQL exceptions/errors Sometimes however, administrators may want eG to disregard certain unhandled exceptions (or handled SQL exceptions), as they may not pose any threat to the stability of the transaction or to the web site/web application. To achieve this, administrators can configure a comma-separated list of such inconsequential

Parameter	Description
	exceptions in the Ignored Exceptions text box. Here, you need to configure each exception you want to exclude using its fully qualified exception class name. For instance, your Excluded Exceptions specification can be as follows: java.sql.SQLException,java.io.FileNotFoundException. **Note that wild card characters cannot be used as part of your specification**. Once the exceptions to be excluded are configured, then this test will exclude all transactions in which such exceptions are captured from its count of Error transactions.
Ignored Characters	**This parameter will appear only if the Advanced Settings flag is set to 'Yes'**. By default, eG excludes all transaction URLs that contain the '\' character from monitoring. If you want eG to ignore transaction URLs with any other special characters, then specify these characters as a comma-separated list in the Ignored Characters text box. For instance, your specification can be: \\,&,~
Max Grouped URLs per Measure Period	**This parameter will appear only if the Advanced Settings flag is set to 'Yes'**. This test groups URLs according to the Max URL Segments specification. These grouped URLs will be the descriptors of the test. For each grouped URL, response time metrics will be aggregated across all transaction URLs in that group and reported. When monitoring web sites/web applications to which the transaction volume is normally high, this test may report metrics for hundreds of descriptors. If all these descriptors are listed in the **Layers** tab page of the eG monitoring console, it will certainly clutter the display. To avoid this, by default, the test displays metrics for a maximum of 50 descriptors – i.e., 50 grouped URLs alone – in the eG monitoring console, during every measure period. This is why, the Max Grouped URLs per measure period parameter is set to 50 by default. To determine which 50 grouped URLs should be displayed in the eG monitoring console, the eG BTM follows the below-mentioned logic: • Top priority is reserved for URL groups with error transactions. This means that eG BTM first scans URL groups for error transactions. If error transactions are found in 50 URL groups, then eG BTM computes the aggregated response time of each of the 50 groups, sorts the error groups in the descending order of their response time, and displays all these 50 groups alone as the descriptors of this test, in the sorted order. • On the other hand, if error transactions are found in only one / a few URL groups – say, only 20 URL groups – then, eG BTM will first arrange these 20 grouped URLs in the descending order of their response time. It will then compute the aggregated response time of the transactions in each of the other groups (i.e., the error-free groups) that were auto-discovered during the same measure period. These other

Parameter	Description
	groups are then arranged in the descending order of the aggregated response time of their transactions. Once this is done, eG BTM will then pick the top-30 grouped URLs from this sorted list. In this case, when displaying the descriptors of this test in the **Layers** tab page, the 20 error groups are first displayed (in the descending order of their response time), followed by the 30 'error-free' groups (also in the descending order of their response time). At any given point in time, you can increase/decrease the maximum number of descriptors this test should support by modifying the value of the Max Grouped URLs per Measure Period parameter.
Max SQl Queries per Transaction	**This parameter will appear only if the Advanced Settings flag is set to 'true'**. Typically, from the detailed diagnosis of a slow/stalled/error transaction on a JVM node, you can drill down to view the SQL queries (if any) executed by that transaction from that node and the execution time of each query. By default, eG picks the first *500* SQL queries executed by the transaction, compares the execution time of each query with the SQL Execution Cutoff configured for this test, and displays only those queries with an execution time that is higher than the configured cutoff. This is why, the Max SQL Queries per Transaction parameter is set to *500* by default. To improve agent performance, you may want the SQL execution cutoff to be compared with the execution time of a less number of queries - say, 200 queries. Similary, to increase the probability of capturing more number of long-running queries, you may want the sql execution cutoff to be compared with the execution time of a large number of queries - say, 1000 queries. For this, you just need to modify the Max SQL Queries per Transaction specification to suit your purpose.
Timeout	By default, the eG agent will wait for 1000 milliseconds for a response from the eG Application Server agent. If no response is received, then the test will timeout. You can change this timeout value, if required.
DD Frequency	Refers to the frequency with which detailed diagnosis measures are to be generated for this test. The default is *1:1*. This indicates that, by default, detailed measures will be generated every time this test runs, and also every time the test detects a problem. You can modify this frequency, if you so desire. Also, if you intend to disable the detailed diagnosis capability for this test, you can do so by specifying *none* against DD frequency.
Detailed Diagnosis	To make diagnosis more efficient and accurate, the eG Enterprise suite embeds an optional detailed diagnostic capability. With this capability, the eG agents can be configured to run detailed, more elaborate tests as and when specific problems are detected. To enable the detailed diagnosis capability of this test for a particular server,

Parameter	Description
	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.

Measures reported by the test:

Measurement	Description	Measurement Unit	Interpretation
All transactions	Indicates the total number of requests received for transactions of this pattern during the last measurement period.	Number	By comparing the value of this measure across transaction patterns, you can identify the most popular transaction patterns. Using the detailed diagnosis of this measure, you can then figure out which specific transactions of that pattern are most requested.
Avg response time	Indicates the average time taken by the transactions of this pattern to complete execution.	Secs	Compare the value of this measure across patterns to isolate the type of transactions that were taking too long to execute. You can then use the detailed diagnosis of the All transactions measure of that group to know how much time each transaction in that group took to execute. This will lead you to the slowest transaction.
Healthy transactions	Indicates the number of healthy transactions of this pattern.	Number	
Healthy transactions percentage	Indicates what percentage of the total number of transactions of this pattern is healthy.	Percent	To know which are the healthy transactions, use the detailed diagnosis of this measure.
Slow transactions	Indicates the number of	Number	This measure will report the number of

Measurement	Description	Measurement Unit	Interpretation
	transactions of this pattern that were slow during the last measurement period.		transactions with a response time higher than the configured Slow Transaction Cutoff (MS). A high value is a cause for concern, as too many slow transactions means that user experience with the web application is poor.
Slow transaction response time	Indicates the average time taken by the slow transactions of this pattern to execute.	Secs	
Slow transactions percentage	Indicates what percentage of the total number of transactions of this pattern is currently slow.	Percent	Use the detailed diagnosis of this measure to know which precise transactions of a pattern are slow. You can drill down from a slow transaction to know what is causing the slowness.
Error transactions	Indicates the number of transactions of this pattern that experienced errors during the last measurement period.	Number	A high value is a cause for concern, as too many error transactions to a web application can significantly damage the user experience with that application.
Error transactions response time	Indicates the average duration for which the transactions of this pattern were processed before an error condition was detected.	Secs	The value of this measure will help you discern if error transactions were also slow.
Error transactions percentage	Indicates what percentage of the total number of transactions of this pattern is experiencing errors.	Percent	Use the detailed diagnosis of this measure to isolate the error transactions. You can even drill down from an error transaction in the detailed diagnosis to determine the cause of the error.
Stalled transactions	Indicates the number of transactions of this pattern that were stalled during the last measurement period.	Number	This measure will report the number of transactions with a response time higher than the configured Stalled Transaction Cutoff (MS). A high value

Measurement	Description	Measurement Unit	Interpretation
			is a cause for concern, as too many stalled transactions means that user experience with the web application is poor.
Stalled transactions response time:	Indicates the average time taken by the stalled transactions of this pattern to execute.	Secs	
Stalled transactions percentage	Indicates what percentage of the total number of transactions of this pattern is stalling.	Percent	Use the detailed diagnosis of this measure to know which precise transactions of a pattern are stalled. You can drill down from a stalled transaction to know what is causing that transaction to stall.
Slow SQL statements executed	Indicates the number of slow SQL queries that were executed by the transactions of this pattern during the last measurement period.	Number	
Slow SQL statement time	Indicates the average execution time of the slow SQL queries that were run by the transactions of this pattern.	Secs	If there are too many slow transactions of a pattern, you may want to check the value of this measure for that pattern to figure out if query execution is slowing down the transactions. Use the detailed diagnosis of the *Slow transactions* measure to identify the precise slow transaction. Then, drill down from that slow transaction to confirm whether/not database queries have contributed to the slowness. Deep-diving into the queries will reveal the slowest queries and their impact on the execution time of the transaction.
Avg CPU time	Indicates the average time for which transactions of this pattern were utilizing	Msecs	Compare the value of this measure across transaction patterns to accurately identify the CPU-intensive

Measurement	Description	Measurement Unit	Interpretation
	the CPU.		transaction patterns. **Note:** This measure is reported only under the following circumstances: • The Enable Thread CPU Monitoring flag of this test is set to **Yes**; • The target application server's JVM implementation supports CPU time monitoring of threads; to verify JVM support for CPU time monitoring, use the procedure described in the Verifying JVM Support for CPU Time and Thread Contention Monitoring.
Avg block time	Indicates the average duration for which transactions of this pattern were blocked and could not execute.	Msecs	If the *Avg response time* for any transaction pattern is very high, you may want to check the value of this measure for that pattern. This will help you figure out whether/not prolonged blocking is causing transactions of that pattern to slow down or stall. **Note:** This measure is reported only under the following circumstances: • The Enable Thread contention Monitoring flag of this test is set to **Yes**; • The target application server's JVM implementation supports contention monitoring of threads; to verify JVM support for thread contention

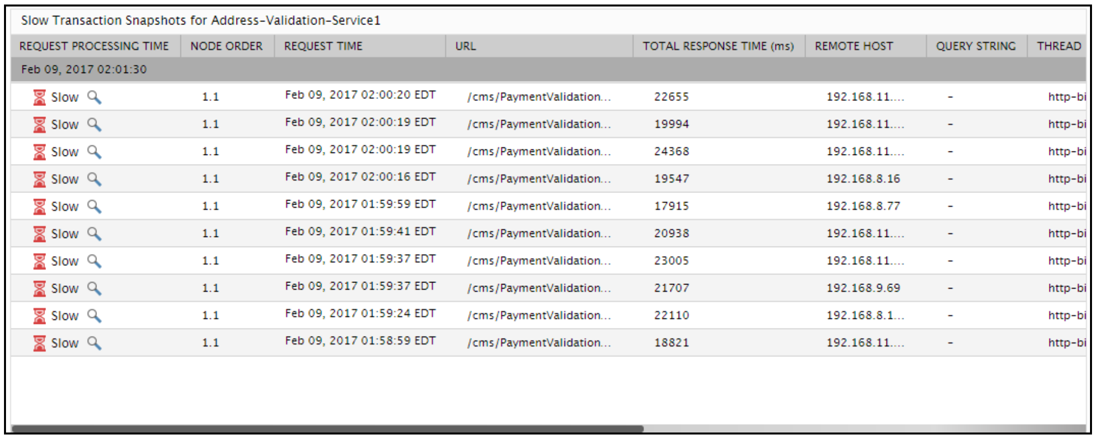
Measurement	Description	Measurement Unit	Interpretation
			monitoring, use the procedure described in the Verifying JVM Support for CPU Time and Thread Contention Monitoring.
Avg wait time	Indicates the average duration for which transactions of this pattern were waiting before they resumed execution.	Msecs	If the *Avg response time* for any transaction pattern is very high, you may want to check the value of this measure for that pattern. This will help you figure out whether/not a very high waiting time is what is causing the transactions to slow down/stall. **Note:** This measure is reported only under the following circumstances: • The Enable Thread Contention Monitoring flag of this test is set to **Yes**; • The target application server's JVM implementation supports contention monitoring of threads; to verify JVM support for thread contention monitoring, use the procedure described in the Verifying JVM Support for CPU Time and Thread Contention Monitoring.
Total transactions per minute	Indicates the number of transactions of this pattern that are executed per minute.	Number	This is a good indicator of the transaction processing ability of the target application server.
Error transactions per minute	Indicates the number of error transactions of this pattern that are executed per minute.	Number	A very low value is desired for this measure. Compare the value of this measure

Measurement	Description	Measurement Unit	Interpretation
			across transaction patterns to find that pattern of transactions that is experiencing errors frequently.

3.3 Detailed Diagnostics

By reporting detailed diagnostics on transaction responsiveness and errors, eG Enterprise not only points you to the slow/stalled/error transaction URLs, but also reveals what could be causing the slowness/errors.

Figure 3.2 reveals detailed diagnosis of the Slow transactions percentage measure of the **Java Business Transactions** test.



Slow Transaction Snapshots for Address-Validation-Service1

REQUEST PROCESSING TIME	NODE ORDER	REQUEST TIME	URL	TOTAL RESPONSE TIME (ms)	REMOTE HOST	QUERY STRING	THREAD
Feb 09, 2017 02:01:30							
Slow	1.1	Feb 09, 2017 02:00:20 EDT	/cms/PaymentValidation...	22655	192.168.11....	-	http-bi
Slow	1.1	Feb 09, 2017 02:00:19 EDT	/cms/PaymentValidation...	19994	192.168.11....	-	http-bi
Slow	1.1	Feb 09, 2017 02:00:19 EDT	/cms/PaymentValidation...	24368	192.168.11....	-	http-bi
Slow	1.1	Feb 09, 2017 02:00:16 EDT	/cms/PaymentValidation...	19547	192.168.8.16	-	http-bi
Slow	1.1	Feb 09, 2017 01:59:59 EDT	/cms/PaymentValidation...	17915	192.168.8.77	-	http-bi
Slow	1.1	Feb 09, 2017 01:59:41 EDT	/cms/PaymentValidation...	20938	192.168.11....	-	http-bi
Slow	1.1	Feb 09, 2017 01:59:37 EDT	/cms/PaymentValidation...	23005	192.168.11....	-	http-bi
Slow	1.1	Feb 09, 2017 01:59:37 EDT	/cms/PaymentValidation...	21707	192.168.9.69	-	http-bi
Slow	1.1	Feb 09, 2017 01:59:24 EDT	/cms/PaymentValidation...	22110	192.168.8.1...	-	http-bi
Slow	1.1	Feb 09, 2017 01:58:59 EDT	/cms/PaymentValidation...	18821	192.168.11....	-	http-bi

Figure 3.2: Detailed diagnosis of the Slow transactions percentage measure of the Java Business Transactions test

The detailed diagnosis reveals the individual transaction URLs in the grouped URL that users requested for, the total response time of each transaction, the client (remote host) from which each transaction request was received, the thread executing the transaction, and the query string of the transaction URL.

The per-transaction response time displayed in Figure 3.2 includes the following:

- The total time for which the transaction request was processed by the target JVM and by other BTM-enabled JVMs in the transaction path thereafter, until the time the response for that transaction request was sent out by the target JVM;

- The time taken by external calls (SQL query / HTTP / JMX / Java / JMS / SAP JCO / async) to other JVMs or backends in the transaction path;

Additionally, the overall experience of the users with each transaction – whether it is slow, stalled, or error - is also revealed in the **REQUEST PROCESSING TIME** column. Furthermore, the HTTP headers, cookies, the HTTP status code returned by the monitored node in response to the transaction request, and the type of HTTP method invoked by the transaction on that node are also revealed. In addition, the following are displayed as part of detailed diagnostics:

- How much time each transaction used the CPU;
- How much time was every transaction blocked;
- How much time did each transaction spend waiting;

CPU-intensive transactions, blocked transactions, and waiting transactions of the chosen pattern can thus be isolated. Also, for the Slow or Stalled transactions, this information will help you determine the probable cause for the transaction slowness - is it because the transactions were blocked for too long and could not execute? or is it because the transactions were waiting for too long a time to continue execution?

The per-transaction statistics are also sorted in the descending order of the transaction response time, starting with the slowest transaction and ending with the healthiest one. In the event that the Avg response time of a grouped URL registers an abnormally high value, you can use these detailed metrics to quickly and accurately identify the exact transaction in the group that is significantly contributing to the poor user experience with the group.

Besides the above, the detailed diagnosis also includes two columns, namely - User Name and Business Context.By default, these two columns will not display any values. This has been done so that administrators can use these columns to display any additional information that they deem useful for troubleshooting transaction slowness. For instance, administrators can configure eG Enterprise to capture the name of the user who initiated each transaction and display the same in the User Name column for every transaction URL in the Detailed Diagnosis page. Likewise, administrators can also tweak eG Enterprise to capture and display information such as fetch type, class name, method name, method signature, session attribute name, URL pattern, etc. against Business Context. Such custom information can also be captured for specific transaction URLs or URL patterns alone. To know how this can be achieved, refer to the Section **3.3.1** topic.

Detailed diagnostics are also available for the Slow transaction percentage, Stalled transaction percentage, and Error transaction percentage measures of the **Java Business Transactions** test.

With the help of these detailed measures, you will be able to quickly and accurately identify the slow, stalled, and error transactions in a grouped URL.

Once a slow/stalled transaction is revealed, the next question is what is causing the transaction to slowdown. Transaction responsiveness can be impacted by any of the following factors:

- An inefficient database query run by the target JVM node;
- In a multi-JVM environment, a time-consuming POJO / non-POJO method called by any JVM node;
- A poorly responsiveness remote service call made by the target JVM node;

With the help of illustrated examples, the links below describe how drill-downs from the detailed diagnostics enable accurate isolation of the root-cause of a transaction slowdown / errors in a transaction.

- Detailed Diagnostics Revealing that an Inefficient Database Query is the Reason for a Slow Transaction
- Detailed Diagnostics Revealing that a Slow JVM Node is Causing Transactions to Slowdown
- Detailed Diagnostics Revealing the Root-cause of an Error Transaction
- Detailed Diagnostics Revealing that a Remote Service Call is the Reason Why a Transaction Slowed Down

3.3.1 Configuring User Name and Business Context

As part of detailed diagnosis, eG BTM displays two columns, namely - User Name and Business Context. By default, these two columns will not display any values. This has been done so that administrators can use these columns to display any additional information that they deem useful for troubleshooting transaction slowness. For instance, administrators can configure eG Enterprise to capture the name of the user who initiated each transaction and display the same in the User Name column for every transaction URL in the Detailed Diagnosis page. Likewise, administrators can also tweak eG Enterprise to capture and display information such as fetch type, class name, method name, method signature, session attribute name, URL pattern, etc. against Business Context. Such custom information can also be captured for specific transaction URLs or URL patterns alone.

To achieve this, follow the steps below:

1. Edit the exclude.props file in the <EG_BTM_INSTALL_DIR>\lib\btm directory.
2. In the file, locate the IC for APM Configuration section.

3. In this section, first create an entry called **IC_IDS** and indicate for how many URLs/URL patterns you want a User Name and/or Business Context to be displayed.

   IC_IDS=1~|~2~|~3~|........|~N

   For instance, if you want to configure a User Name and/or Business Context to be displayed for 4 patterns of URLs. then your IC_IDS specification will be as follows:

   IC_IDS=1~|~2~|~3~|~4

4. Next, append an **IC** entry for every URL/URL pattern for which a User Name and/or a Business Context is to be displayed. Each IC specification should be configured in the following format:

   IC_<<URLIndex>>=<<Entry Description>>~|~<<Field Type>>~|~<<Fetch Type>>~|~<<Fully Qualified Class Name>>~|~<<Method Name>>~|~<<Method Signature>>~|~<<Method Argument Index>>~|~<<Fetch Once>>~|~<<Pass Request Object>>~|~<<Execute at Start of the Transaction>>~|~<<Session Attribute Name>>~|~<<Matching URL Pattern>>

5. Let us take a look at each variable in the specification. *<<URLIndex>>* refers to the serial number that identifies the URL/URL pattern to which the IC specification applies. This can be any number, depending upon the total number of URLs/URL patterns for which IC specifications need to be defined. For instance, if you want to display a user name and/or business context for 3 URLs/URL patterns, then, you will have to insert three separate IC specifications here, each with the *<<URLIndex>>* 1, 2, and 3, respectively, as shown below.

   IC_1=
   IC_2=
   IC_3=

   The *<<URLIndex>>* has to be sequentially incremented, as and when a new IC specification is appended.

6. *<<Entry Description>>* can be a text string or a keyword that uniquely identifies the URL/URL pattern to which the IC specification applies. For instance, if you are configuring an IC specification for the URL pattern, */sports*, then you can configure *sports* as the *<<Entry Description>>*.

7. <<*Field Type*>> should indicate whether you want to fetch a User Name for the URL/URL pattern or a Business Context. The <<*Field Type*>> should be either 1 or 2, where 1 denotes User Name and 2 denotes Business Context.

8. <<*Fetch Type*>> refers to the approach using which you want to capture the User Name or Business Context for a transaction URL. eG Enterprise prescribes three approaches to capturing the same:

   - Method Argument - If you have written a Java method that takes input arguments for capturing the user name or business context, then use the Method Argument approach.
   - Static Method - A Static method belongs to the class and not to the object(instance). A static method can access only static data. It cannot access non-static data (instance variables). A static method can call only other static methods and can not call a non-static method from it. If you have written a static Java method for retrieving the user name or business context, then use this approach.
   - Session Attribute - If the User Name or Business Context is available as a session attribute, then you can configure the eG agent to access that session attribute and retrieve the required information.

   The rest of the specification will change based on the approach you choose. In the specification, type 1 to choose the Method Argument approach, 2 to choose the Static Method approach, and 3 to pick the Session Attribute approach.

9. <<*Fully Qualified Class Name*>> is applicable only if the <<Fetch Type>> is either 1 or 2. If so, then specify the name of the class that contains the method definitions for fetching the User Name or Business Context. If a packaged class is to be used, then specify the fully qualified class name - eg., *com/samples/MyProgram*. If its not a packaged class, then simply specify the class name - eg., *MyProgram*.

10. <<*Method Name*>> refers to the method that should be invoked for capturing the User Name or Business Context. Specify the name of that method here.

11. <<*Method Signature*>> is applicable only if the <<Fetch Type>> is 1. If the <<*Method Name*>> configured takes one/more input arguments, then specify a comma-separated list of such arguments in the place of <<*Method Signature*>>. Typically, if these input arguments are of primitive type, then you can specify them as is . However, if they are objects or wrapper classes, then they should be specified using the fully qualified object name or wrapper class name - eg., */java/lang/String*.

    On the other hand, if the <<*Method Name*>> configured does not take any input parameters, then enter *null* here.

12. <<*Method Argument Index*>> is applicable only if <<*Fetch Type*>> is 1. If the <<*Fetch Type*>> is 2 or 3, then replace <<*MethodArgumentIndex*>> with *none*. For <<*Fetch Type*>> 1, in the place of <<*Method Argument Index*>>, you need to define at what position of the method invocation call, the information you need - i.e., whether User Name or Business Context - resides. For instance, if the method invocation call is *Method1(int userID, string userName)*, then to fetch the User Name, the <<*Method Argument Index*>> will be 2. This is because, the second argument, *string userName*, is the one that fetches the User Name information.
13. <<*Fetch Once*>> takes the value true or false. This is presently not handled. Nevertheless, you need to set it to either true or false. The value you set will not in any way impact the functionality or the output of the method.
14. <<*Pass Request Object*>> is applicable only if <<*FetchType*>> is 2 - i.e., the static method approach. The static method that you use to fetch the User Name or Business Context should either take only one request object or no request objects. If the static method you use uses a single request object, enter *true* in the place of <<*Pass Request Object*>>. If the method you have written does not use any request objects, then specify *false*. If the <<*Fetch Type*>> is 1 or 3, then enter *none*.
15. <<*Execute at Start of Trans*>> is applicable only if <<*Fetch Type*>> is 2 or 3. If <<*Fetch Type*>> is 1, then set this flag to *none*. On the other hand, if << *Fetch Type*>> is 2 or 3, then set the value of this flag to *true* or *false*. For example, say, you want to fetch the value of User Name from a session attribute and have hence set <<*Fetch Type*>> to 3. Let's say that the session attribute captures the user name only when the transaction is in progress; not when it begins. In this case, you have to set the <<*Execute at Start of Trans*>> to *false*, so that the user name is obtained from the session attribute towards the end of the transaction. On the other hand, if the session attribute captures the user name at the beginning of the transaction, set this flag to *true*. In this case, the user name is obtained from the session attribute at the start of the transaction itself.
16. <<*Session Attribute Name*>> is applicable only if <<*Fetch Type*>> is 3. If so, then specify the name of the session attribute from which the User Name or Business Context has to be fetched.
17. <<*URL Pattern*>> is applicable to all <<*Fetch Types*>>. Here, specify a comma-separated list of URLs or URL patterns for which the User Name or Business Context has to be captured and displayed as part of detailed diagnosis. The URL patterns can include wild card patterns - eg., **/WebPoc*,*Web**
18. A sample IC specification is as follows:

*IC_*
1=MethodParams~|~1~|~1~|~com/egi/poc/UserName~|~isValidUser~|~null~|~1~|~true~|~false~|~tru
e~|~null~|~/WebPoc**

Finally, save the file.

19. Then, restart the application.

Note:

Some of the limitations of each of the approaches are as follows:

Method Argument Approach

- Not possible to get the argument values of predefined methods.
- Not possible to capture all the argument values of a method.

Static Method Approach

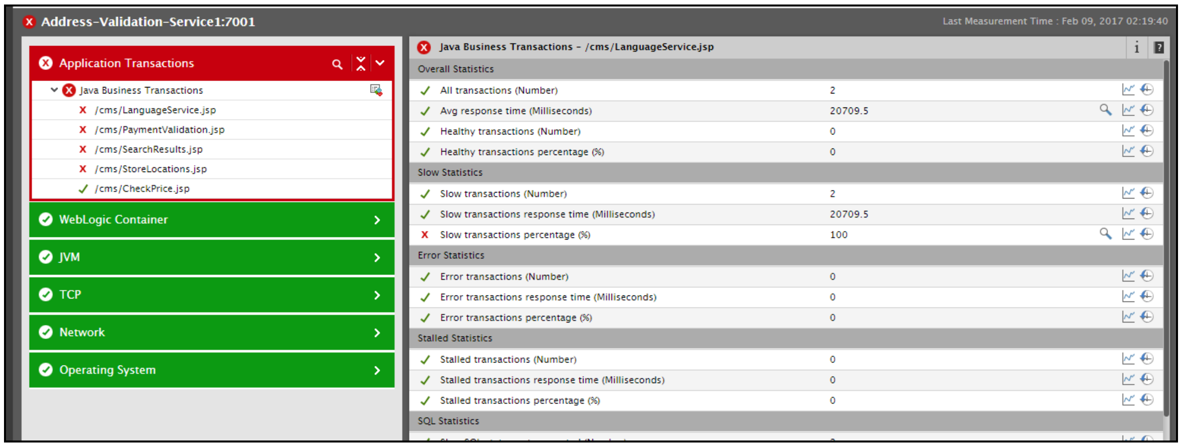
- Not possible to get the return value of private static methods.
- Not possible to get the output of static methods that have more than one parameter.

Session Attribute Approach

Not possible to get the session value if the response is committed at the end of the transaction.

3.3.2 Detailed Diagnostics Revealing that an Inefficient Database Query is the Reason for a Slow Transaction

Let us consider the example of a web application that has been deployed on the Oracle WebLogic server, Address-Validation-Service1:7001. Users of the web application complained that every time they tried to browse the LanguageService web page on the web application, the response was very poor. Using eG's Java Business Transactions test of the Oracle WebLogic server,Address-Validation-Service1:7001, you can promptly capture this anomaly! As you can see in Figure 1 below, the Java Business Transactions test has accurately captured and reported that the Slow transactions percentage for the /cms/LanguageService.jsp is 100%. This means that 100% of the requests for the LanguageService.jsp transaction were serviced slowly (see Figure 3.3)!


Address-Validation-Service1:7001

Last Measurement Time : Feb 09, 2017 02:19:40

Application Transactions

- Java Business Transactions
  - /cms/LanguageService.jsp
  - /cms/PaymentValidation.jsp
  - /cms/SearchResults.jsp
  - /cms/StoreLocations.jsp
  - /cms/CheckPrice.jsp

WebLogic Container

JVM

TCP

Network

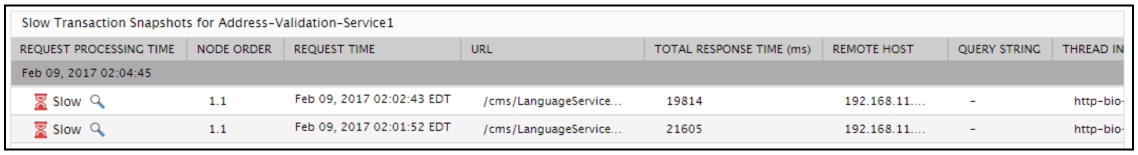
Operating System

Java Business Transactions - /cms/LanguageService.jsp

Overall Statistics		
✓	All transactions (Number)	2
✓	Avg response time (Milliseconds)	20709.5
✓	Healthy transactions (Number)	0
✓	Healthy transactions percentage (%)	0
Slow Statistics		
✓	Slow transactions (Number)	2
✓	Slow transactions response time (Milliseconds)	20709.5
X	Slow transactions percentage (%)	100
Error Statistics		
✓	Error transactions (Number)	0
✓	Error transactions response time (Milliseconds)	0
✓	Error transactions percentage (%)	0
Stalled Statistics		
✓	Stalled transactions (Number)	0
✓	Stalled transactions response time (Milliseconds)	0
✓	Stalled transactions percentage (%)	0
SQL Statistics		

Figure 3.3: The Layers tab page indicating that all requests for /Easykart/PaymentPage.jsp were slow

To know which request received the slowest response, click the **DIAGNOSIS** icon against the Slow transactions percentage measure in 3.3.2. Figure 3.4 will then appear listing all the transaction requests that were slow, the time at which each request was sent, the total response time of every request, the client from which the request was received, the query string of the transaction URL, and more.


Slow Transaction Snapshots for Address-Validation-Service1

REQUEST PROCESSING TIME	NODE ORDER	REQUEST TIME	URL	TOTAL RESPONSE TIME (ms)	REMOTE HOST	QUERY STRING	THREAD IN
Feb 09, 2017 02:04:45							
Slow	1.1	Feb 09, 2017 02:02:43 EDT	/cms/LanguageService...	19814	192.168.11....	-	http-bio-
Slow	1.1	Feb 09, 2017 02:01:52 EDT	/cms/LanguageService...	21605	192.168.11....	-	http-bio-

Figure 3.4: Detailed Diagnosis of the Slow transactions percentage measure

Since the requests are arranged in the descending order of their response time, a quick look at the detailed diagnostics will lead you to the precise request that is the slowest. But, why is response to this request slow? To answer this question, click the 'magnifying glass' icon against Slow in n the slowest request (i.e., the topmost request in Figure 3.4).

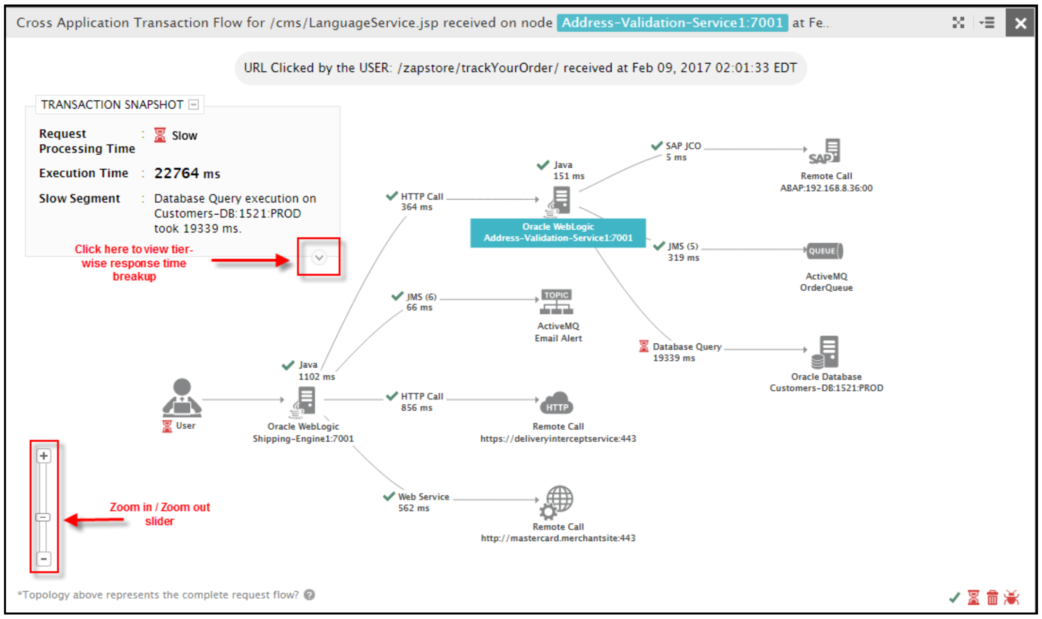
Figure 3.5 will then appear revealing the cross-application flow of the slow transaction. This flow diagram clearly reveals the following:

- The JVMs and backends through which the transaction travelled;
- The time for which the transaction request was processed at each BTM-enabled JVM; **note that this time will not be computed for JVMs that are in the transaction path, but are not BTM-enabled and those that are BTM-enabled but are not managed by eG;**
- The exit calls made by each BTM-enabled node to another node as part of the transaction's journey, the time consumed by each exit call,and the number of times each type of call was made;

the following exit calls are supported by eG BTM:

- Database Query
- HTTP
- Web service
- JMS
- LDAP
- RMI
- Java mail API
- EJB
- JSF (Java Server Faces)
- Runtime

Note that the EJB exit call is supported only for JBoss, WebLogic, and WebSphere nodes.


Cross Application Transaction Flow for /cms/LanguageService.jsp received on node Address-Validation-Service1:7001 at Fe..

URL Clicked by the USER: /zapstore/trackYourOrder/ received at Feb 09, 2017 02:01:33 EDT

TRANSACTION SNAPSHOT

Request Processing Time	:	Slow
Execution Time	:	22764 ms
Slow Segment	:	Database Query execution on Customers-DB:1521:PROD took 19339 ms.

*Topology above represents the complete request flow?

Click here to view tier-wise response time breakup
Zoom in / Zoom out slider
User
Java 1102 ms
Oracle WebLogic Shipping-Engine1:7001
HTTP Call 364 ms
JMS (6) 66 ms
ActiveMQ Email Alert
HTTP Call 856 ms
Remote Call https://deliveryinterceptservice:443
Web Service 562 ms
Remote Call http://mastercard.merchantsite:443
Java 151 ms
Oracle WebLogic Address-Validation-Service1:7001
SAP JCO 5 ms
Remote Call ABAP:192.168.8.36:00
JMS (5) 319 ms
ActiveMQ OrderQueue
Database Query 19339 ms
Oracle Database Customers-DB:1521:PROD
TOPIC
QUEUE
HTTP
SAP

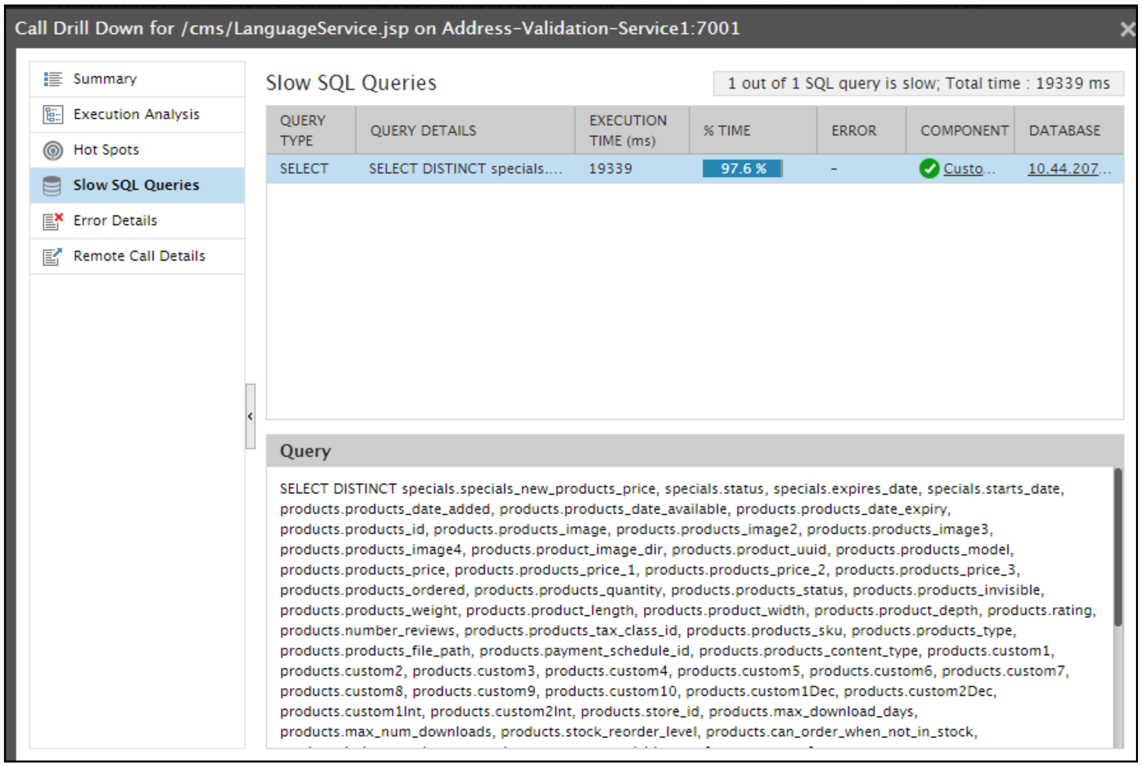
Figure 3.5: Cross-application transaction flow

Note:

- If a BTM-enabled node appears 'grayed out' in the cross-application transaction flow, it denotes that eG BTM could not collect detailed diagnostics for that node. The reasons for this could be either or both of the following:
    - Transaction responsiveness on the 'grayed out' node was either healthy or was only slightly slow, and hence, did not appear in the list of Top-N slow transactions.
    - Slow data transmission from eG agent to manager;
- If a JVM node makes a JMS call to a messaging server, then. in the transaction topology, that messaging server will be identified by the name of the messaging service provider and the name of the exact queue/topic that is managing the JMS request. If a JVM node makes a SQL query call on a database server, then the details displayed for that database server in the transaction topology depends upon whether/not that database server is managed by eG Enterprise. If the database server is not managed by eG Enterprise, then such a database server will be represented in the topology using the server type (whether Oracle, Microsoft SQL etc.) and the name of the database that was accessed by the SQL query. To know the IP and port number of the unmanaged database server, you can drill-down from the **Database queries**call in the topology. On the other hand, if the database server in question is being monitored by eG Enterprise, then such a server will be represented in the topology using the server type, nick name, port number, and the database name. Additionally, the SID will be displayed in case of an Oracle database server, and the instance name will be displayed in case of an instance-based Microsoft SQL server.
- EJB calls from a client and to a server on the same host will not be captured by eG BTM, and will hence not be displayed in the cross-application transaction topology.
- Sometimes, empty nodes – i.e., nodes without any details – will be visible in the cross-application transaction flow topology. Likewise, the time spent on certain external calls may also not be displayed in the topology. This is owing to inconsistencies in the collection of detailed diagnostics.

Using conventional color codes and intuitive icons, the transaction flow chart precisely pinpoints where the transaction slowed down. In the case of Figure 3.5 above, from the color-coding it is clear that the **Database Query** executed by the Oracle WebLogic server – Address-Validation-Service1:7001 - is taking a long time for execution. The question now is which query is this. To determine that, click on **Database Query** in Figure 3.5.

Drilling down from **Database Query** in Figure 3.5 automatically opens the list of **SQL Queries** executed by the slow transaction in question (see Figure 3.6). The execution time of each query and what percentage of the total response time of the transaction each query is consuming will be displayed here. From Figure 3.6, it is evident that a **SELECT DISTINCT specials. . .** query is taking over 19000 milliseconds for execution – this is apparently 97% of the total response time of the target transaction. This time-consuming query is what is causing the transaction to slow down. To view the complete query, click on that query in the **SQL Queries** list of Figure 3.6. The detailed query will then be displayed in the **Query** section of Figure 3.6.


Call Drill Down for /cms/LanguageService.jsp on Address-Validation-Service1:7001
Summary
Execution Analysis
Hot Spots
Slow SQL Queries
Error Details
Remote Call Details
Slow SQL Queries
1 out of 1 SQL query is slow; Total time : 19339 ms
Query

QUERY TYPE	QUERY DETAILS	EXECUTION TIME (ms)	% TIME	ERROR	COMPONENT	DATABASE
SELECT	SELECT DISTINCT specials....	19339	97.6 %	-	✓ Custo...	10.44.207...

```
SELECT DISTINCT specials.specials_new_products_price, specials.status, specials.expires_date, specials.starts_date,
products.products_date_added, products.products_date_available, products.products_date_expiry,
products.products_id, products.products_image, products.products_image2, products.products_image3,
products.products_image4, products.product_image_dir, products.product_uuid, products.products_model,
products.products_price, products.products_price_1, products.products_price_2, products.products_price_3,
products.products_ordered, products.products_quantity, products.products_status, products.products_invisible,
products.products_weight, products.product_length, products.product_width, products.product_depth, products.rating,
products.number_reviews, products.products_tax_class_id, products.products_sku, products.products_type,
products.products_file_path, products.payment_schedule_id, products.products_content_type, products.custom1,
products.custom2, products.custom3, products.custom4, products.custom5, products.custom6, products.custom7,
products.custom8, products.custom9, products.custom10, products.custom1Dec, products.custom2Dec,
products.custom1Int, products.custom2Int, products.store_id, products.max_download_days,
products.max_num_downloads, products.stock_reorder_level, products.can_order_when_not_in_stock,
```

Figure 3.6: Analyzing the slow query

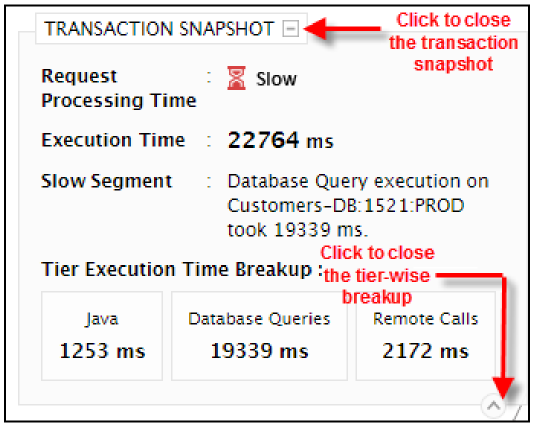
This way, using a short sequence of mouse clicks, you have zeroed-in on the source of the transaction slowness.

The **TRANSACTION SNAPSHOT** section in Figure 3.5 leads you to the same root-cause, **without requiring any clicks!** The details provided by this section are as follows:

- **User Experience:** The user experience with the LanguageService transaction; in our example, this is Slow
- **Execution Time:** The total response time of the LanguageServicetransaction;
- **Slow Segment:** Where exactly the LanguageService transaction slowed down;

From the **Slow Segment** display, it is evident that a database query executed by the LanguageService.jsp transaction on the Customers-DB database took over 19000 millisecs for execution, thereby slowing down the entire transaction! This corroborates our findings from the cross-application transaction flow and the subsequent query analysis.

Now, click on the down-arrow button at the bottom tip of the **TRANSACTION SNAPSHOT** section (as indicated by Figure 3.5). Doing so will reveal a tier-wise breakup of the transaction response time (see Figure 3.7). This way, you can quickly compare response time across tiers, and accurately isolate where the bottleneck lies – in this case, it is in the database queries.


TRANSACTION SNAPSHOT

Request Processing Time : Slow

Execution Time : 22764 ms

Slow Segment : Database Query execution on Customers-DB:1521:PROD took 19339 ms.

Tier Execution Time Breakup :

Java	Database Queries	Remote Calls
1253 ms	19339 ms	2172 ms

Click to close the transaction snapshot
Click to close the tier-wise breakup

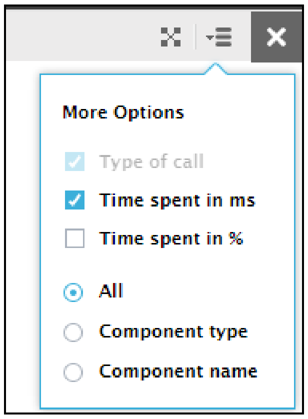
Figure 3.7: Tier-wise response time breakup

To close the tier-wise breakup, click on the up arrow button indicated by Figure 3.7.

You can even close the transaction snapshot pop-up if you want to by clicking on the ⊟ button alongside the title **TRANSACTION SNAPSHOT** (as indicated by Figure 3.7).

Let us now revisit the cross-application flow diagram of the LanguageService transaction. You can use the top-down slider at the bottom, left corner of the flow diagram (as indicated by Figure 3.5) to zoom your diagram in and out.

Moreover, by default, the time spent by the transaction at every point cut is reported in milliseconds in the flow diagram. You can reconfigure the flow diagram to express the time spent as a percentage of total transaction response time instead. For this, first click the button at the right, top corner of the flow diagram. The options depicted by 3.3.2 will then appear.


More Options

- [✓] Type of call
- [✓] Time spent in ms
- [ ] Time spent in %

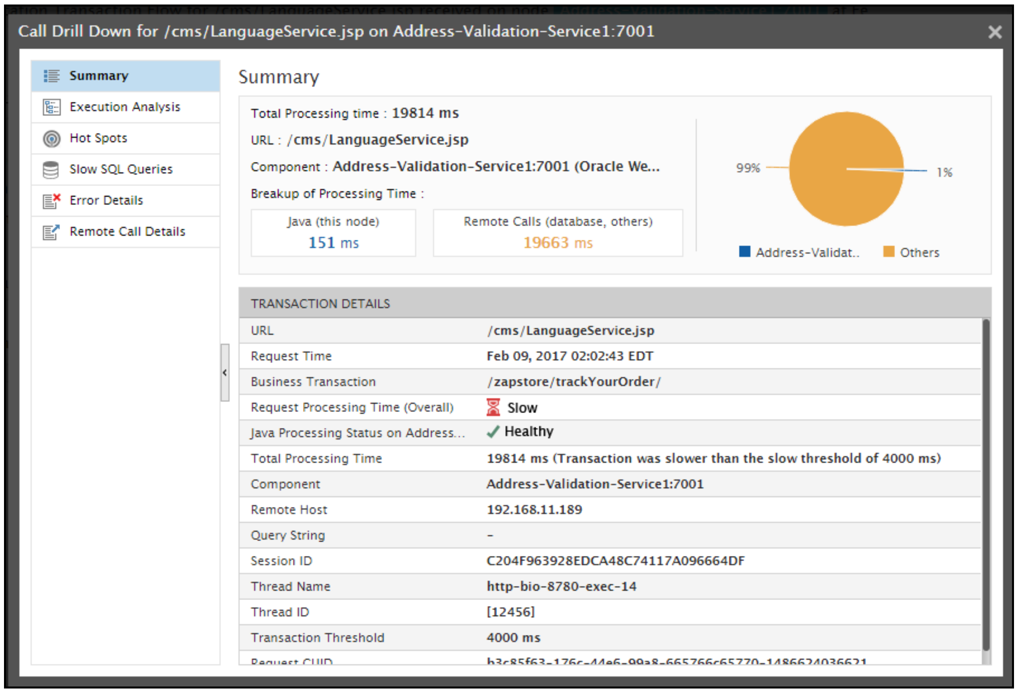
- [✓ All]
- Component type
- Component name

Figure 3.8: Expressing the time spent at every point cut as a percentage of total transaction response time

Uncheck the **Time spent in ms** check box in 3.3.2 and select the **Time spent in %** check box to make sure that the response time at every point cut is displayed as a percentage of total transaction response time. The percentage will enable you to better judge where the transaction spent maximum time.

You can also choose the **Component type** or **Component name** options in 3.3.2 to have the component type only or the component name only (as the case may be) displayed for each of the nodes in the cross-application transaction flow. By default, both component type and name will be displayed for each node.

Let us now explore the **Summary** section of the call drill down. For that, click the **Summary** option in the left panel of Figure 3.6. Figure 3.9 will appear.



Call Drill Down for /cms/LanguageService.jsp on Address-Validation-Service1:7001

- Summary
- Execution Analysis
- Hot Spots
- Slow SQL Queries
- Error Details
- Remote Call Details

Summary

Total Processing time : 19814 ms

URL : /cms/LanguageService.jsp

Component : Address-Validation-Service1:7001 (Oracle We...

Breakup of Processing Time :

Java (this node)	Remote Calls (database, others)
151 ms	19663 ms

TRANSACTION DETAILS

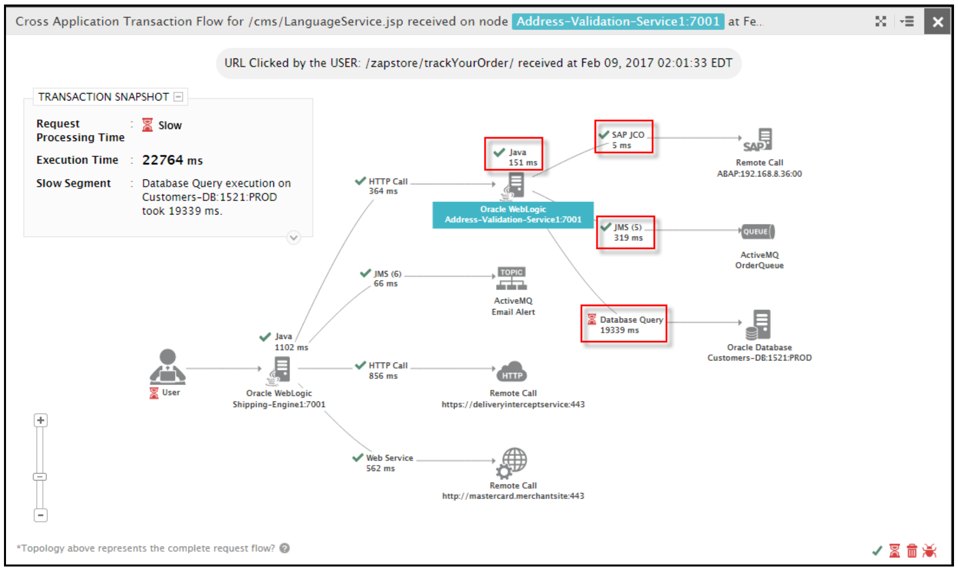
URL	/cms/LanguageService.jsp
Request Time	Feb 09, 2017 02:02:43 EDT
Business Transaction	/zapstore/trackYourOrder/
Request Processing Time (Overall)	Slow
Java Processing Status on Address...	✓ Healthy
Total Processing Time	19814 ms (Transaction was slower than the slow threshold of 4000 ms)
Component	Address-Validation-Service1:7001
Remote Host	192.168.11.189
Query String	-
Session ID	C204F963928EDCA48C74117A096664DF
Thread Name	http-bio-8780-exec-14
Thread ID	[12456]
Transaction Threshold	4000 ms
Request GUID	b3c85f63-176c-44e6-99a8-665766c65770-1486624036621

Figure 3.9: A summary of the performance of the JVM node, Address-Validation-Service1:7878

The **Summary** section provides a quick summary of the performance of the monitored transaction, LanguageService.jsp, on the JVM node that executed the slow database query – i.e., the Oracle WebLogic server, Address-Validation-Service1:7001. .

From the **Summary**, you can infer that the LanguageService transaction was processed for a total of 19814 milliseconds on Address-Validation-Service1:7001. If you take a look at the transaction topology now (see Figure 3.10), you will be able to understand that this processing time is the sum of the following:

- The time for which the transaction was processed internally by the Address-Validation-Service1:7001 server – 151 ms
- The time taken by Address-Validation-Service1:7001 to execute a database query for the transaction and retrieve results – 19339 ms
- The time taken by Address-Validation-Service1:7001 to make a JMS call to a messaging server and pull data from the message queue OrderQueue – 319 ms
- The time taken by Address-Validation-Service1:7001 to make a SAP JCO call to a SAP server – 5 ms


Cross Application Transaction Flow for /cms/LanguageService.jsp received on node Address-Validation-Service1:7001 at Fe..

URL Clicked by the USER: /zapstore/trackYourOrder/ received at Feb 09, 2017 02:01:33 EDT

TRANSACTION SNAPSHOT

Request Processing Time	:	Slow
Execution Time	:	22764 ms
Slow Segment	:	Database Query execution on Customers-DB:1521:PROD took 19339 ms.

User
Oracle WebLogic Shipping-Engine1:7001
Java 1102 ms
HTTP Call 364 ms
JMS (6) 66 ms
HTTP Call 856 ms
Web Service 562 ms
Oracle WebLogic Address-Validation-Service1:7001
Java 151 ms
SAP JCO 5 ms
JMS (5) 319 ms
Database Query 19339 ms
ActiveMQ Email Alert
Remote Call https://deliveryinterceptservice:443
Remote Call http://mastercard.merchantsite:443
Remote Call ABAP:192.168.8.36:00
ActiveMQ OrderQueue
Oracle Database Customers-DB:1521:PROD
*Topology above represents the complete request flow?

Figure 3.10: How the total processing time of the transaction on Address-Validation-Service1:7001 is computed

The **Breakup of Processing Time** section in Figure 3.9 clearly indicates how the **Total Processing time** is computed. From this section, you can also glean where the slowdown originated – within the JVM node? Or when making external calls from the JVM node? In the case of our example, the problem is with the remote calls.

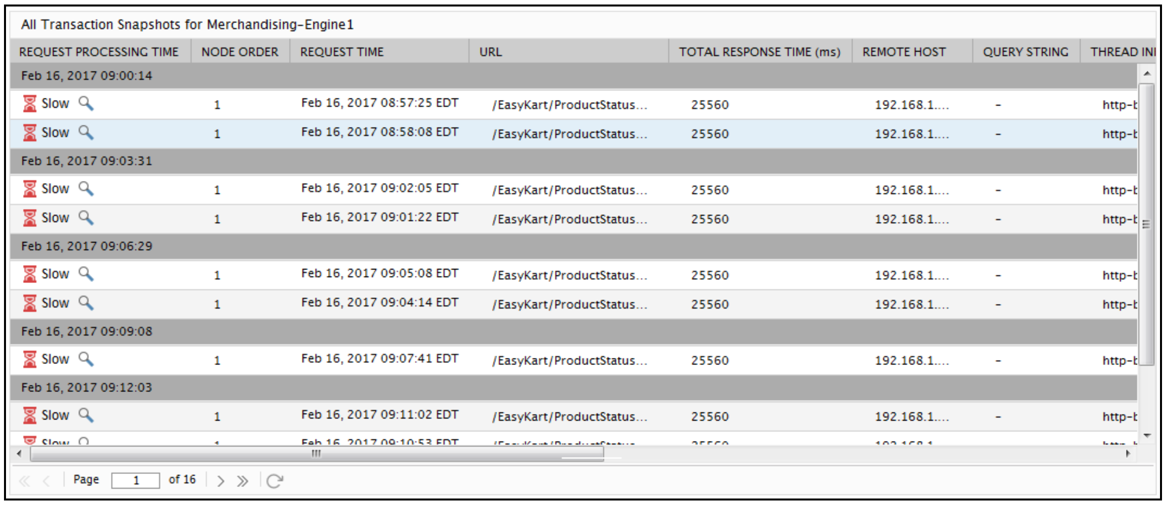
Next, take a look at the **URL** displayed in the **Summary** section. As you can see, the **URL** is that of the **Business Transaction** that was zoomed into - i.e., LanguageService.jsp. However, sometimes, while the **Business Transaction** may continue to be LanguageService.jsp, the **URL** could be different. This is because, the **URL** refers to the URL that was hit when an HTTP call is made by one JVM node to another. This means that when accessing the LanguageService.jsp web page on Address-Validation-Service1:7001, if that web page had hit another URL, then that URL will be displayed against **URL**.

Additionally, the **Summary** section also reports the **Query String** of the URL, the **Session ID** of the session in which the transaction is processed on the Address-Validation-Service1:7878 server, and **Thread Name** of the thread that processed the transaction.

The **Summary** section also differentiates between the overall **User Experience** of a transaction and the **Java Processing Status** of that transaction on a particular JVM node. In the case of our example, the **Summary** section clearly reveals that the **User Experience** of the transaction is Slow. At the same time, eG has also detected that the slowness did not occur because of a processing bottleneck on the Address-Validation-Service1:7001 server. This is why, eG maintains that the **Java Processing Status of the Address-Validation-Service1:7001** server is Healthy.

3.3.3 Detailed Diagnostics Revealing that a Slow JVM Node is Causing Transactions to Slowdown

Let us consider the example of a web application, where the following transactions are slow.



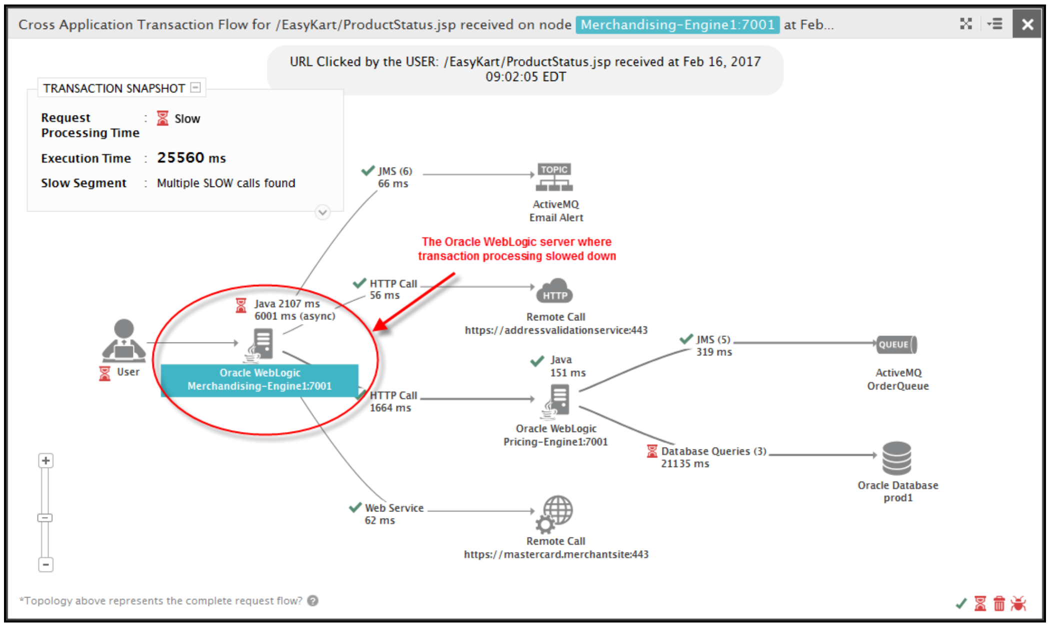
All Transaction Snapshots for Merchandising-Engine1

REQUEST PROCESSING TIME	NODE ORDER	REQUEST TIME	URL	TOTAL RESPONSE TIME (ms)	REMOTE HOST	QUERY STRING	THREAD INI
Feb 16, 2017 09:00:14							
Slow	1	Feb 16, 2017 08:57:25 EDT	/EasyKart/ProductStatus...	25560	192.168.1....	-	http-t
Slow	1	Feb 16, 2017 08:58:08 EDT	/EasyKart/ProductStatus...	25560	192.168.1....	-	http-t
Feb 16, 2017 09:03:31							
Slow	1	Feb 16, 2017 09:02:05 EDT	/EasyKart/ProductStatus...	25560	192.168.1....	-	http-t
Slow	1	Feb 16, 2017 09:01:22 EDT	/EasyKart/ProductStatus...	25560	192.168.1....	-	http-t
Feb 16, 2017 09:06:29							
Slow	1	Feb 16, 2017 09:05:08 EDT	/EasyKart/ProductStatus...	25560	192.168.1....	-	http-t
Slow	1	Feb 16, 2017 09:04:14 EDT	/EasyKart/ProductStatus...	25560	192.168.1....	-	http-t
Feb 16, 2017 09:09:08							
Slow	1	Feb 16, 2017 09:07:41 EDT	/EasyKart/ProductStatus...	25560	192.168.1....	-	http-t
Feb 16, 2017 09:12:03							
Slow	1	Feb 16, 2017 09:11:02 EDT	/EasyKart/ProductStatus...	25560	192.168.1....	-	http-t
Slow	1	Feb 16, 2017 09:10:53 EDT	/EasyKart/ProductStatus...	25560	192.168.1....	-	http-t

Page 1 of 16

Figure 3.11: Detailed diagnosis of the Avg response time measure

Let us focus on the slow /Easykart/ProductStatus.jsp in Figure 3.11. To zoom into the transaction, click on it. The flow of the ProductStatus.jsp transaction will then be displayed as depicted by Figure 3.12.



Cross Application Transaction Flow for /EasyKart/ProductStatus.jsp received on node Merchandising-Engine1:7001 at Feb...

URL Clicked by the USER: /EasyKart/ProductStatus.jsp received at Feb 16, 2017 09:02:05 EDT

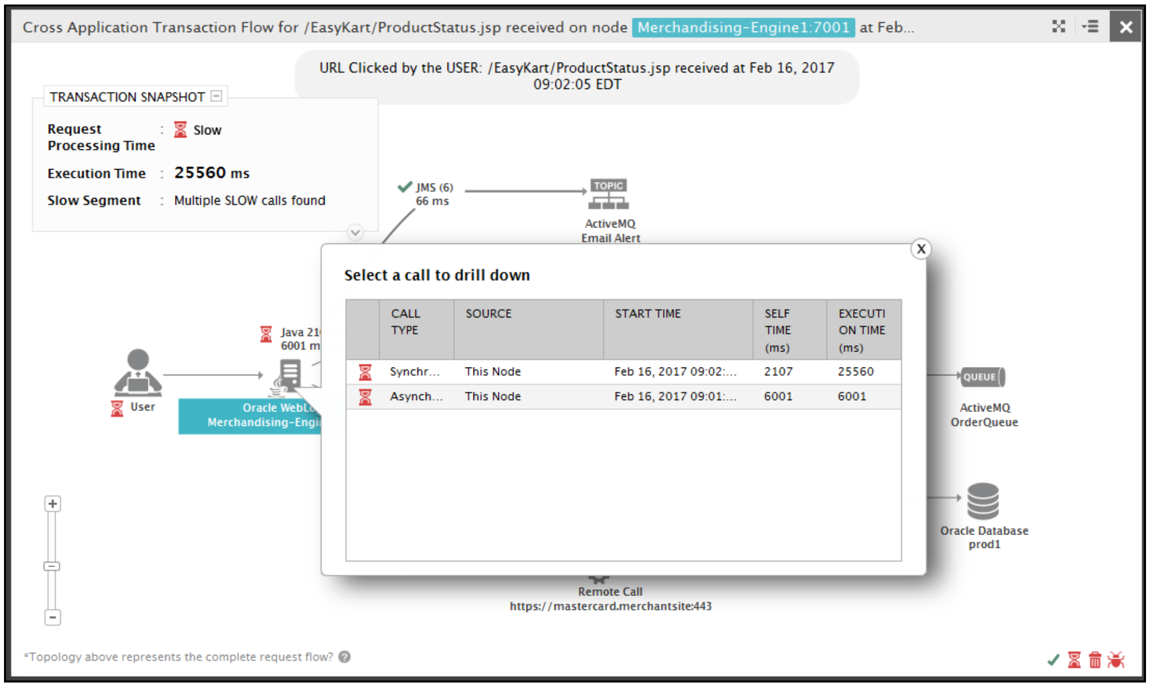
TRANSACTION SNAPSHOT

Request Processing Time	: Slow
Execution Time	: 25560 ms
Slow Segment	: Multiple SLOW calls found

*Topology above represents the complete request flow?

Figure 3.12: The cross-application flow of the ProductStatus.jsp transaction

From the transaction flow, it is evident that the transaction slowed down on the Oracle WebLogic server, Merchandising-Engine1:7001. The question now is what type of processing on the Oracle WebLogic server delayed the transaction in question. A closer look at the WebLogic server icon in Figure 3.12 will answer this question as well! As indicated by Figure 3.12, the Merchandising-Engine1:7001 server processed Java methods synchronously for 2107 milliseconds and asynchronously for over 6000 milliseconds. Comparing the two execution times points the needle of suspicion towards the synchronous Java call made by the WebLogic server. If so, which exact Java method is slowing down the transaction? To identify the same, let us zoom into the WebLogic server by clicking on it in Figure 3.12. An intermediate window depicted by Figure 3.13 will then appear.


Cross Application Transaction Flow for /EasyKart/ProductStatus.jsp received on node Merchandising-Engine1:7001 at Feb...

URL Clicked by the USER: /EasyKart/ProductStatus.jsp received at Feb 16, 2017 09:02:05 EDT

TRANSACTION SNAPSHOT

Request Processing Time	:	Slow
Execution Time	:	25560 ms
Slow Segment	:	Multiple SLOW calls found

Select a call to drill down

	CALL TYPE	SOURCE	START TIME	SELF TIME (ms)	EXECUTI ON TIME (ms)
⌛	Synchr...	This Node	Feb 16, 2017 09:02:...	2107	25560
⌛	Asynch...	This Node	Feb 16, 2017 09:01:...	6001	6001

*Topology above represents the complete request flow?

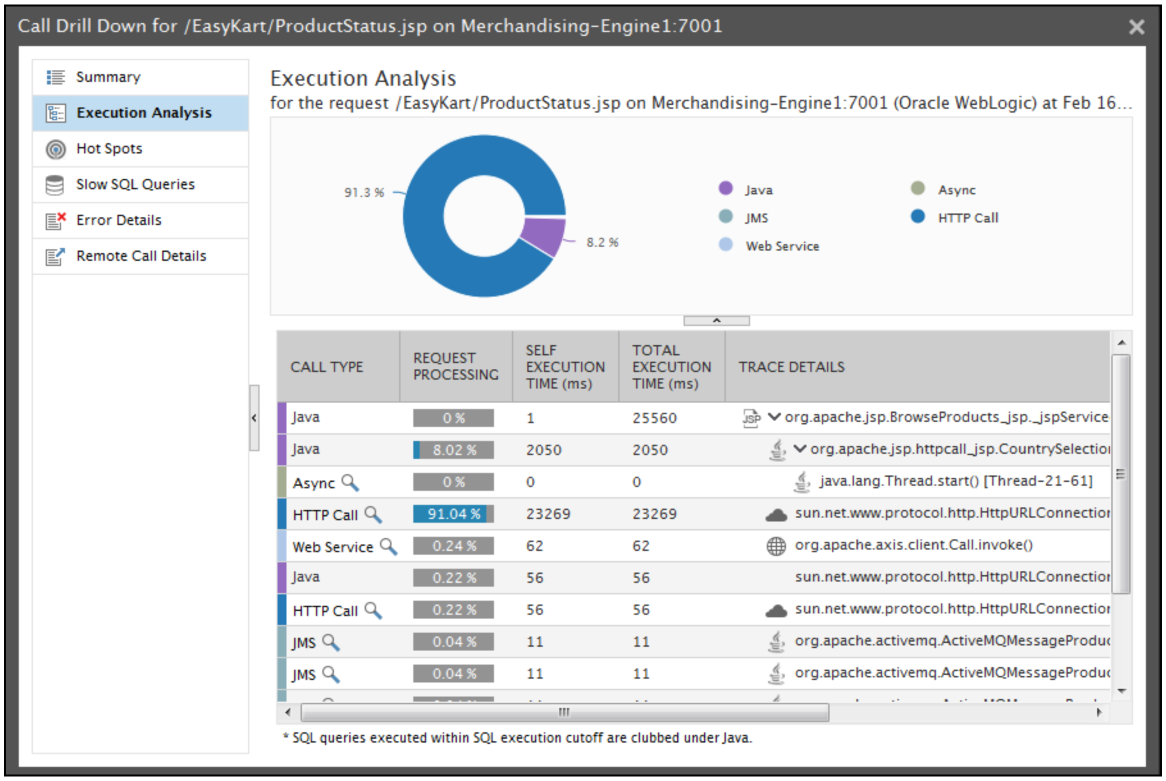
Figure 3.13: An intermediate modular window

This intermediate window will appear only under the following circumstances:

- If a node receives and processes multiple synchronous / asynchronous requests from one/more external sources; and/or
- If one/more asynchronous threads are invoked by a node in response to requests to it;

Typically, from this window, you will be able to quickly determine the number of synchronous and asynchronous calls that a particular JVM node processed. In the case of our example, we can clearly infer from the intermediate window that the Merchandising-Engine1:7001 server executed a synchronous and an asynchronous call.

For each synchronous and asynchronous call, this window will also display the self execution time and total execution time of that call. Self execution time is the time it took for the synchronous/asynchronous call to perform Java processing alone. Total execution time is the time taken by the synchronous/asynchronous call to perform both Java and non-Java (eg., HTTP, Database, etc.) processing. By comparing the self and total execution times across calls, you will be able to accurately identify the exact call that took too long to execute, the call type, and whether such a call was slow in processing Java or non-Java. Accordingly, we can clearly deduce from the intermediate window of Figure 3.13 that the synchronous call made by Merchandising-Engine1:7001 server in our example performed Java processing for a much longer time than desired. To be able to precisely identify the exact Java method that caused the delay, click on the synchronous call in Figure 3.13. Figure 3.14 will then appear.



Execution Analysis

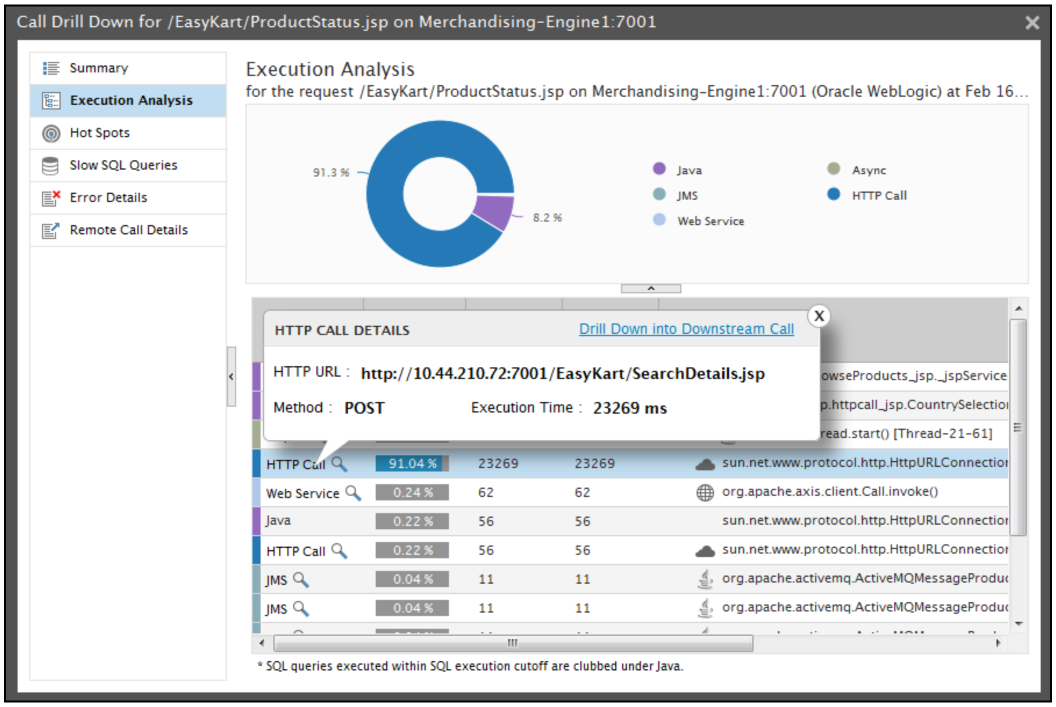
for the request /EasyKart/ProductStatus.jsp on Merchandising-Engine1:7001 (Oracle WebLogic) at Feb 16...

CALL TYPE	REQUEST PROCESSING	SELF EXECUTION TIME (ms)	TOTAL EXECUTION TIME (ms)	TRACE DETAILS
Java	0 %	1	25560	org.apache.jsp.BrowseProducts_jsp._jspService
Java	8.02 %	2050	2050	org.apache.jsp.httpcall_jsp.CountrySelectio
Async	0 %	0	0	java.lang.Thread.start() [Thread-21-61]
HTTP Call	91.04 %	23269	23269	sun.net.www.protocol.http.HttpURLConnectio
Web Service	0.24 %	62	62	org.apache.axis.client.Call.invoke()
Java	0.22 %	56	56	sun.net.www.protocol.http.HttpURLConnectio
HTTP Call	0.22 %	56	56	sun.net.www.protocol.http.HttpURLConnectio
JMS	0.04 %	11	11	org.apache.activemq.ActiveMQMessageProdu
JMS	0.04 %	11	11	org.apache.activemq.ActiveMQMessageProdu

* SQL queries executed within SQL execution cutoff are clubbed under Java.

Figure 3.14: The call graph of the synchronous call

Figure 3.14 provides a detailed **Execution Analysis** of the synchronous call. As part of this analysis, a pie chart is presented that quickly reveals the percentage of time the WebLogic server in our example spent processing the server's Java code and making external JMS / SAP JCO / SQL query calls. The table below the pie chart in Figure 3.14 lists the exact methods that performed Java processing or made the remote calls. A quick look at this table reveals that the Java method, org.apache.jsp.BrowseProducts_jsp_jspService..., invoked a series of child methods and external calls, which together took over 25000 milliseconds to execute. However, the method itself took only around 1 millisecond to execute (self execution time)! Browsing the child methods called by the parent method reveals that the transaction spent over 90% of its time on the HTTP call, 'sun.net.www.protocol..'. This means that the 'sun.net.www.protocol..' is the method that is delaying the BrowseProducts.jsp transaction. To know the exact URL that this HTTP call hit, move your mouse pointer over the call in Figure 3.14.



Execution Analysis
for the request /EasyKart/ProductStatus.jsp on Merchandising-Engine1:7001 (Oracle WebLogic) at Feb 16...

HTTP CALL DETAILS

HTTP URL : http://10.44.210.72:7001/EasyKart/SearchDetails.jsp
Method : POST Execution Time : 23269 ms

				owseProducts_jsp._jspService
				p.httpcall_jsp.CountrySelectio
				read.start() [Thread-21-61]
HTTP Call	91.04 %	23269	23269	sun.net.www.protocol.http.HttpURLConnectio
Web Service	0.24 %	62	62	org.apache.axis.client.Call.invoke()
Java	0.22 %	56	56	sun.net.www.protocol.http.HttpURLConnectio
HTTP Call	0.22 %	56	56	sun.net.www.protocol.http.HttpURLConnectio
JMS	0.04 %	11	11	org.apache.activemq.ActiveMQMessageProdu
JMS	0.04 %	11	11	org.apache.activemq.ActiveMQMessageProdu

* SQL queries executed within SQL execution cutoff are clubbed under Java.

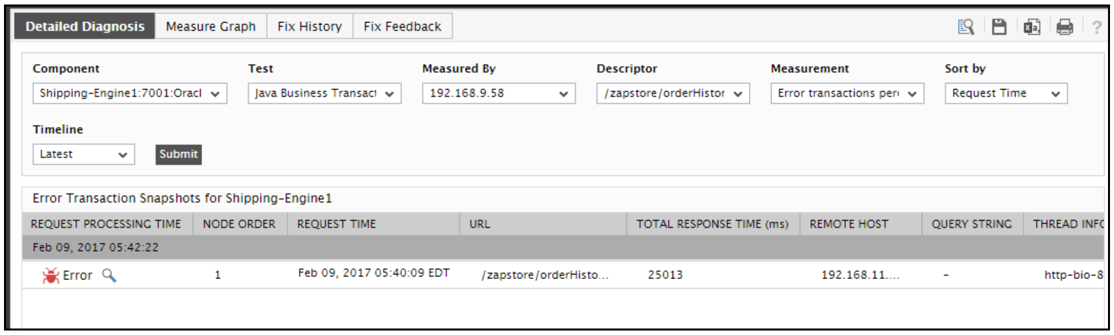
Figure 3.15: The URL hit by an HTTP call

Figure 3.15 will then appear revealing that the /EasyKart/SearchDetails.jsp was the URL that was hit by the HTTP call.

This way, eG BTM enables you to diagnose the root-cause of slowness in your synchronous and asynchronous calls using just a few mouse clicks!

3.3.4 Detailed Diagnostics Revealing the Root-cause of an Error Transaction

The detailed diagnosis of the Error transactions measure reveals the complete URLs of the error transactions of a particular business transaction pattern. The total response time of each error transaction and the time at wihich every such transaction was requested can be ascertained from the detailed diagnosis. To zoom into the nature of the error and where it occurred, click on the 'magnifying glass' icon against the corresponding 'Error' icon in the **TRANSACTION USER EXPERIENCE** column of Figure 3.16.



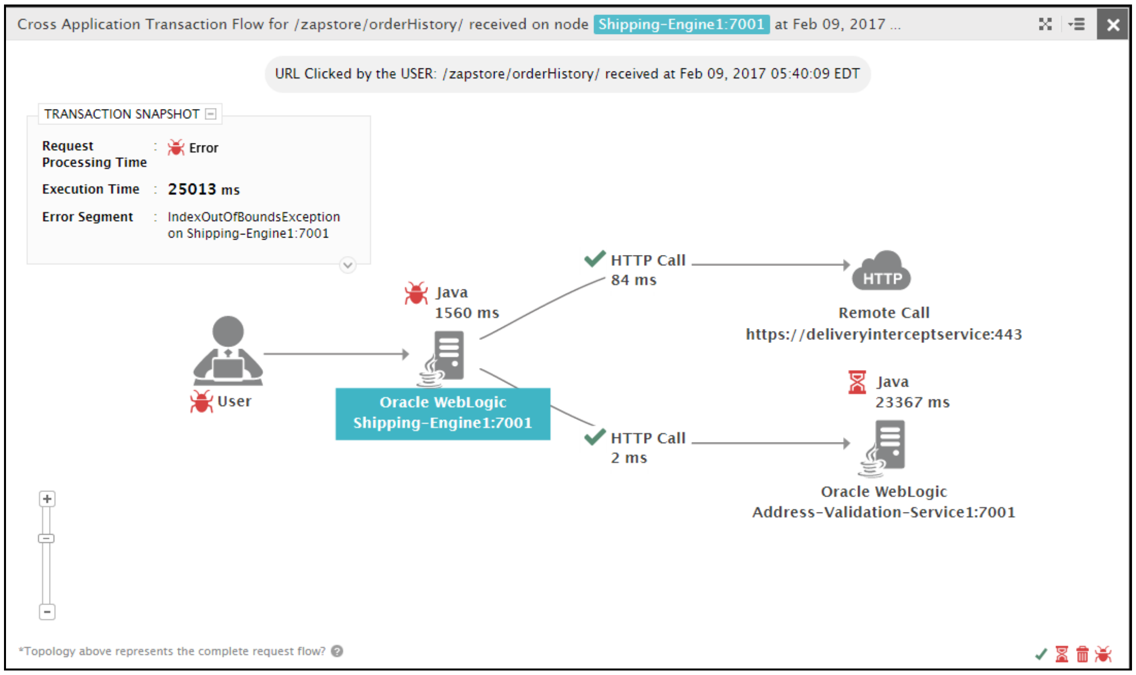
Component: Shipping-Engine1:7001:Oracl
Test: Java Business Transact
Measured By: 192.168.9.58
Descriptor: /zapstore/orderHistor
Measurement: Error transactions per
Sort by: Request Time
Timeline: Latest

Error Transaction Snapshots for Shipping-Engine1

REQUEST PROCESSING TIME	NODE ORDER	REQUEST TIME	URL	TOTAL RESPONSE TIME (ms)	REMOTE HOST	QUERY STRING	THREAD INFO
Feb 09, 2017 05:42:22							
Error	1	Feb 09, 2017 05:40:09 EDT	/zapstore/orderHisto...	25013	192.168.11....	-	http-bio-8

Figure 3.16: The detailed diagnosis of the Error transactions measure

3.3.4 will then appear, which will chart the entire path of the error transaction end-to-end. Using conventional color-codes, this visual representation will accurately pinpoint where the error has occurred.


Cross Application Transaction Flow for /zapstore/orderHistory/ received on node Shipping-Engine1:7001 at Feb 09, 2017 ...

URL Clicked by the USER: /zapstore/orderHistory/ received at Feb 09, 2017 05:40:09 EDT

TRANSACTION SNAPSHOT

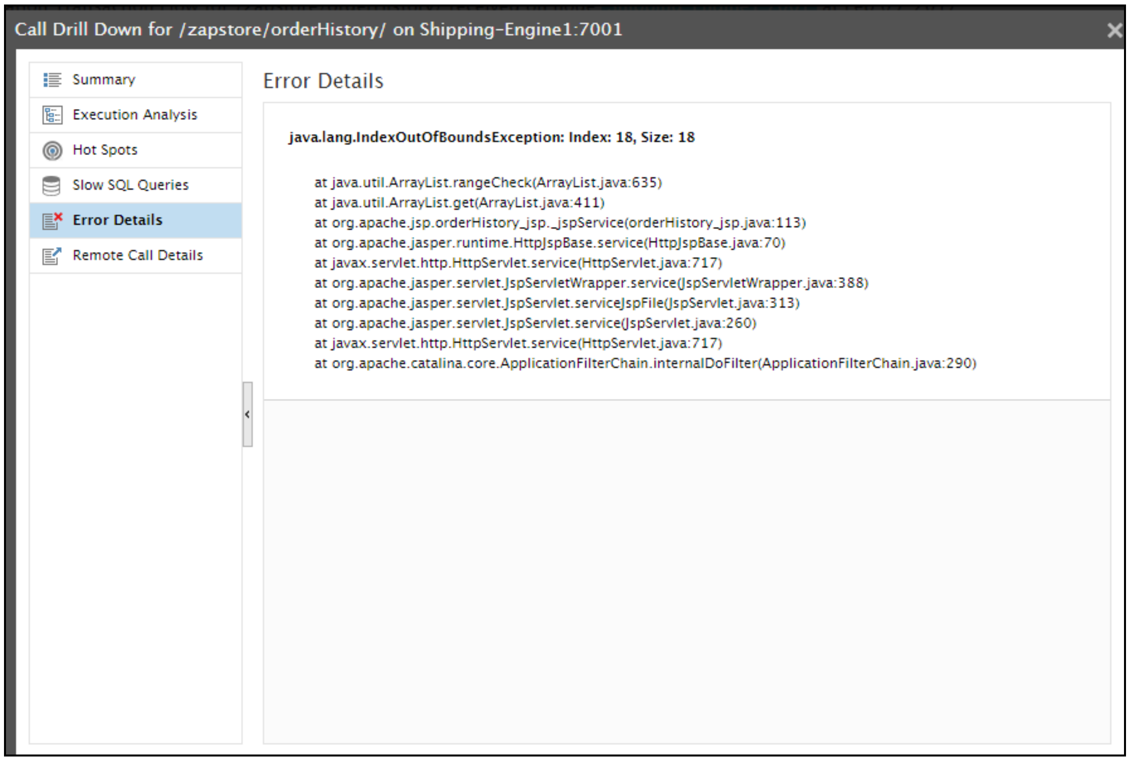
Request Processing Time : Error
Execution Time : 25013 ms
Error Segment : IndexOutOfBoundsException on Shipping-Engine1:7001

*Topology above represents the complete request flow?

Figure 3.17: The error transaction path revealing where the error has occurred

In the example of 3.3.4 above, the error seems to have occurred on the Shipping-Engine1:7001 (Oracle WebLogic) server being monitored. To know what the error is, click on the Shipping-Engine1:7001 server in 3.3.4.

Figure 3.18 that appears next opens an **Error Details** section, which displays the complete details of the error.

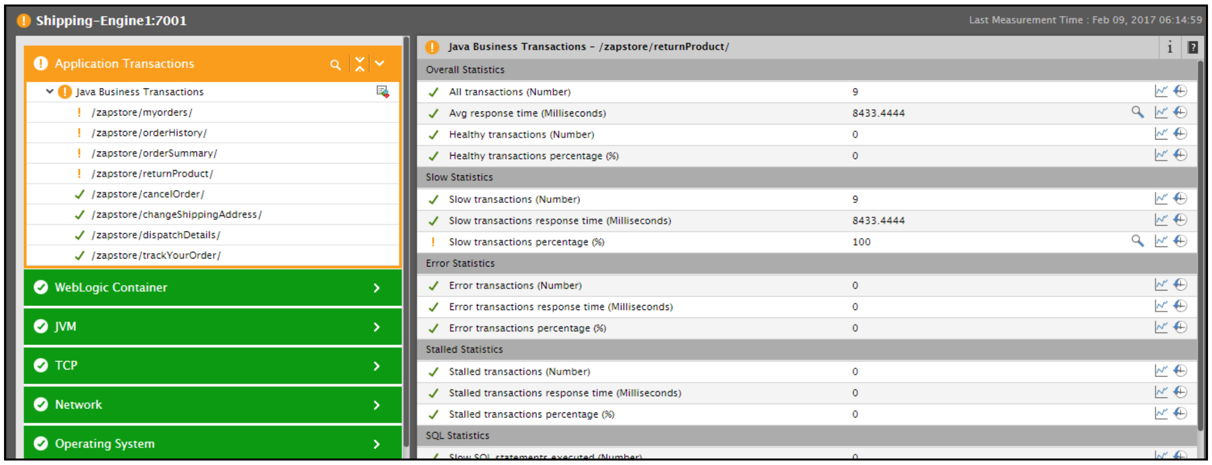


```
java.lang.IndexOutOfBoundsException: Index: 18, Size: 18

    at java.util.ArrayList.rangeCheck(ArrayList.java:635)
    at java.util.ArrayList.get(ArrayList.java:411)
    at org.apache.jsp.orderHistory_jsp._jspService(orderHistory_jsp.java:113)
    at org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:70)
    at javax.servlet.http.HttpServlet.service(HttpServlet.java:717)
    at org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:388)
    at org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:313)
    at org.apache.jasper.servlet.JspServlet.service(JspServlet.java:260)
    at javax.servlet.http.HttpServlet.service(HttpServlet.java:717)
    at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:290)
```

Figure 3.18: Error details

3.3.5 Detailed Diagnostics Revealing that a Remote Service Call is the Reason Why a Transaction Slowed Down

According to Figure 3.19 below, slowness has been detected in 9 transactions of the pattern, /Easykart/Login.jsp. To know the exact URLs of the slow transactions, click on the ‘magnifying glass’ icon against Slow transactions in Figure 3.19.



Shipping-Engine1:7001

Last Measurement Time : Feb 09, 2017 06:14:59

Application Transactions

- Java Business Transactions
  - ! /zapstore/myorders/
  - ! /zapstore/orderHistory/
  - ! /zapstore/orderSummary/
  - ! /zapstore/returnProduct/
  - ✓ /zapstore/cancelOrder/
  - ✓ /zapstore/changeShippingAddress/
  - ✓ /zapstore/dispatchDetails/
  - ✓ /zapstore/trackYourOrder/

WebLogic Container

JVM

TCP

Network

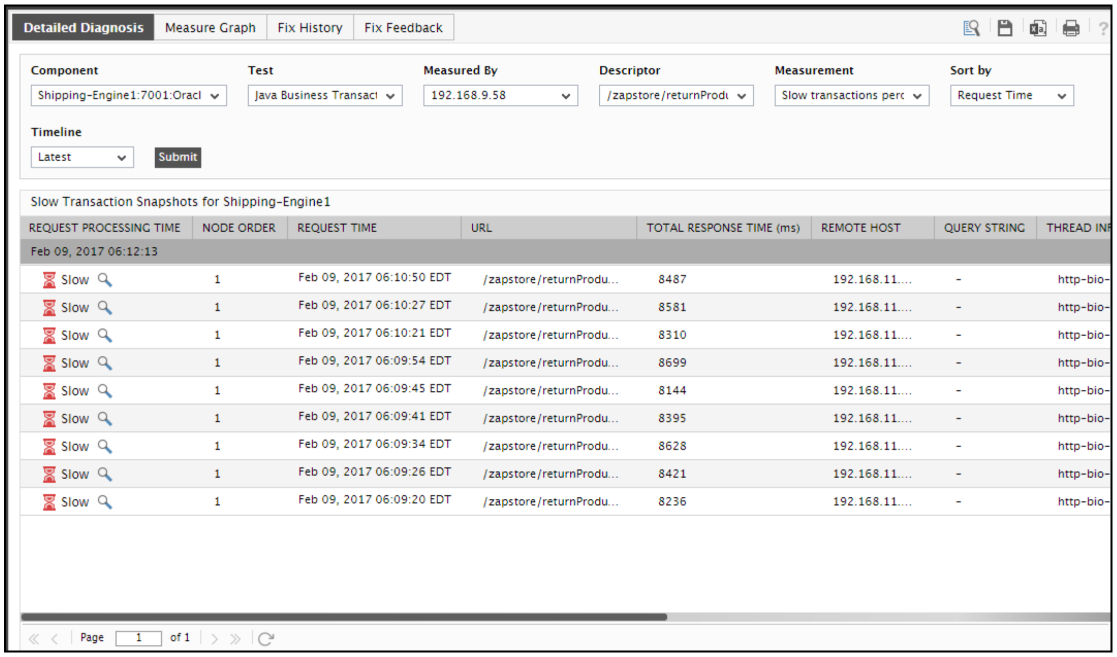
Operating System

Java Business Transactions - /zapstore/returnProduct/

Overall Statistics		
✓	All transactions (Number)	9
✓	Avg response time (Milliseconds)	8433.4444
✓	Healthy transactions (Number)	0
✓	Healthy transactions percentage (%)	0
Slow Statistics		
✓	Slow transactions (Number)	9
✓	Slow transactions response time (Milliseconds)	8433.4444
!	Slow transactions percentage (%)	100
Error Statistics		
✓	Error transactions (Number)	0
✓	Error transactions response time (Milliseconds)	0
✓	Error transactions percentage (%)	0
Stalled Statistics		
✓	Stalled transactions (Number)	0
✓	Stalled transactions response time (Milliseconds)	0
✓	Stalled transactions percentage (%)	0
SQL Statistics		
✓	Slow SQL statements executed (Number)	0

Figure 3.19: The Layers tab page revealing that 100% of the transactions of the pattern /zapstore/returnProduct are slow

Figure 3.20 will then appear listing the slow transactions URLs. To drill down to the source of the slowness of any of these transactions, click on the ‘magnifying glass’ icon alongside the ‘Slow’ icon of that transaction.



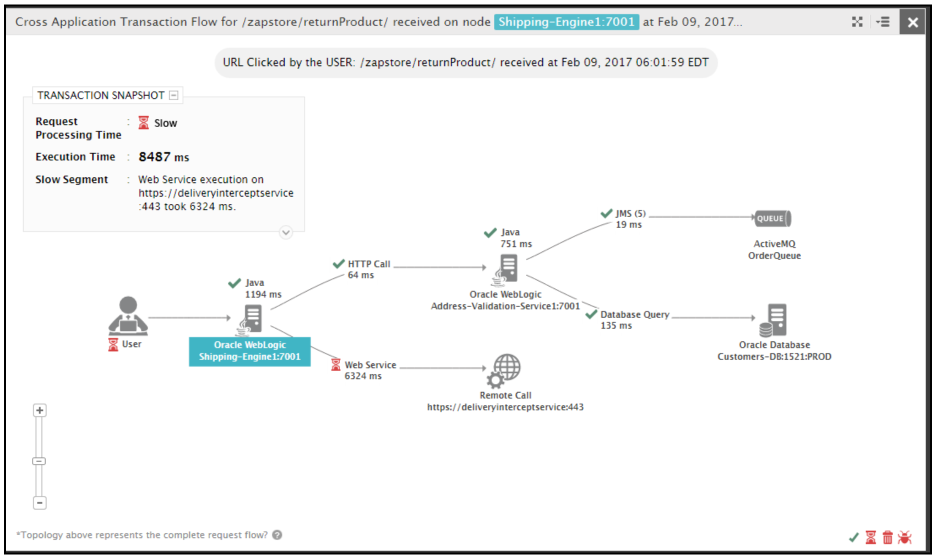
Component: Shipping-Engine1:7001:Oracl
Test: Java Business Transact
Measured By: 192.168.9.58
Descriptor: /zapstore/returnProdu
Measurement: Slow transactions perc
Sort by: Request Time
Timeline: Latest

Slow Transaction Snapshots for Shipping-Engine1

REQUEST PROCESSING TIME	NODE ORDER	REQUEST TIME	URL	TOTAL RESPONSE TIME (ms)	REMOTE HOST	QUERY STRING	THREAD INF
Feb 09, 2017 06:12:13							
Slow	1	Feb 09, 2017 06:10:50 EDT	/zapstore/returnProdu...	8487	192.168.11....	-	http-bio-
Slow	1	Feb 09, 2017 06:10:27 EDT	/zapstore/returnProdu...	8581	192.168.11....	-	http-bio-
Slow	1	Feb 09, 2017 06:10:21 EDT	/zapstore/returnProdu...	8310	192.168.11....	-	http-bio-
Slow	1	Feb 09, 2017 06:09:54 EDT	/zapstore/returnProdu...	8699	192.168.11....	-	http-bio-
Slow	1	Feb 09, 2017 06:09:45 EDT	/zapstore/returnProdu...	8144	192.168.11....	-	http-bio-
Slow	1	Feb 09, 2017 06:09:41 EDT	/zapstore/returnProdu...	8395	192.168.11....	-	http-bio-
Slow	1	Feb 09, 2017 06:09:34 EDT	/zapstore/returnProdu...	8628	192.168.11....	-	http-bio-
Slow	1	Feb 09, 2017 06:09:26 EDT	/zapstore/returnProdu...	8421	192.168.11....	-	http-bio-
Slow	1	Feb 09, 2017 06:09:20 EDT	/zapstore/returnProdu...	8236	192.168.11....	-	http-bio-

Figure 3.20: Detailed diagnosis listing the slow transactions of the pattern /zapstore/returnProduct

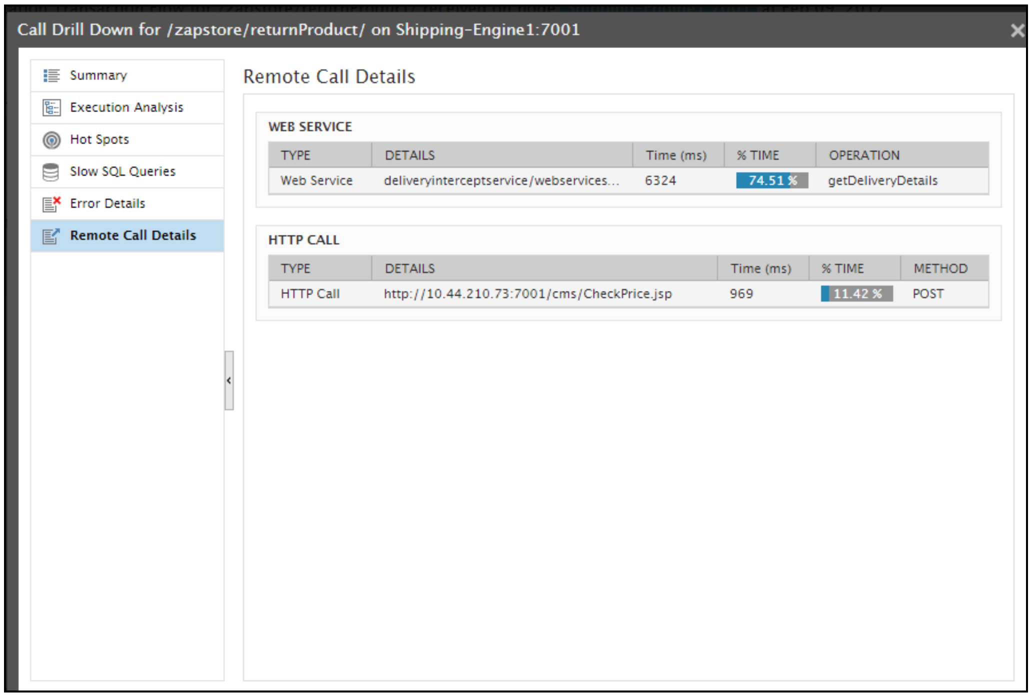
Figure 3.21 will then appear depicting how the transaction flows. From Figure 3.21, it is clear that a Web service call made by the Oracle WebLogic server, Shipping-Engine1:7001, to a delivery intercept service in the backend is slowing down the transaction.


Cross Application Transaction Flow for /zapstore/returnProduct/ received on node Shipping-Engine1:7001 at Feb 09, 2017...
URL Clicked by the USER: /zapstore/returnProduct/ received at Feb 09, 2017 06:01:59 EDT
TRANSACTION SNAPSHOT
User
Java 1194 ms
Oracle WebLogic Shipping-Engine1:7001
HTTP Call 64 ms
Java 751 ms
Oracle WebLogic Address-Validation-Service1:7001
JMS (5) 19 ms
QUEUE
ActiveMQ OrderQueue
Database Query 135 ms
Oracle Database Customers-DB:1521:PROD
Web Service 6324 ms
Remote Call https://deliveryinterceptservice:443
*Topology above represents the complete request flow?

Request Processing Time	:	Slow
Execution Time	:	8487 ms
Slow Segment	:	Web Service execution on https://deliveryinterceptservice :443 took 6324 ms.

Figure 3.21: Cross-application transaction flow depicting that the problem is with the Web Service call

To know more about this call, click the Web Service icon in Figure 3.21. A **Remote Call Details** window will then open listing all the remote calls made by the Shipping-Engine1:7001 server. From this window you can infer that the Web Service call made to the delivery intercept service is consuming nearly 75% of the transaction execution time. As you can see, a few quick mouse clicks from a Slow transaction in Figure 3.20 has lead you to the precise web service call that is delaying the transaction.


Call Drill Down for /zapstore/returnProduct/ on Shipping-Engine1:7001

Remote Call Details

WEB SERVICE

TYPE	DETAILS	Time (ms)	% TIME	OPERATION
Web Service	deliveryinterceptservice/webservices...	6324	74.51 %	getDeliveryDetails

HTTP CALL

TYPE	DETAILS	Time (ms)	% TIME	METHOD
HTTP Call	http://10.44.210.73:7001/cms/CheckPrice.jsp	969	11.42 %	POST

Figure 3.22: List of remote service calls made by the Shipping-Engine1:7001 server

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.