Monitoring Java Applications

eG Innovations Product Documentation

www.eginnovations.com

Total Performance Visibility

Table of Contents

Table of Figures

Chapter 1: Introduction

Java applications are predominantly used in enterprises today owing to their multi-platform nature. Once written, a Java application can be run on heterogeneous platforms with no additional configuration. This is why, the Java technology is widely used in the design and delivery of many critical web and non-web-based applications.

The prime concern of the administrators of these applications is knowing how well the application is functioning, and how to troubleshoot issues (if any) in the performance of these applications. Most web application server vendors prescribe custom APIs for monitoring – for instance, WebSphere and WebLogic allow administrators to use their built-in APIs for performance monitoring and problem detection. The details of these APIs and how eG Enterprise uses them to monitor the application server in question are discussed elaborately in this document.

Chapter 2: Pre-requisites for Monitoring Java Applications

The Java Application model that eG Enterprise prescribes provides both *agentless* and *agent-based* monitoring support to Java applications. The eG agent, deployed either on the application host or on a remote Windows host in the environment (depending upon the monitoring approach – whether agent-based or agentless), can be configured to connect to the JRE used by the application and pull out metrics of interest, using either of the following methodologies:

- JMX (Java Management Extensions)
- SNMP (Simple Network Management Protocol)

Note:

The eG agent uses the specifications prescribed by JSR 174 to perform JVM monitoring.

This is why, each test mapped to the top 2 layers of Figure 1 provides administrators with the option to pick a monitoring **MODE** - i.e., either **JMX** or **SNMP**. The remaining test configuration depends upon the mode chosen.

Since both JMX and SNMP support are available for JRE 1.5 and above only, the *Java Application* model can be used to monitor **only those applications that are running JRE 1.5 and above**. The supported JVMs are as follows:

- Sun JVM 1.5 or higher
- JRockit JVM 5.0 R27.1 or higher
- IBM JRE 1.5 or higher
- OpenJDK 1.5 or higher
- Azul Zing JVM 1.6 or higher

The sections to come discuss how to enable JMX and SNMP for JRE.

2.1 Enabling JMX Support for JRE

In older versions of Java (i.e., JDK/JRE 1.1, 1.2, and 1.3), very little instrumentation was built in, and custom-developed byte-code instrumentation had to be used to perform monitoring. From JRE/JDK 1.5 and above however, support for Java Management Extensions (JMX) were pre-built into JRE/JDK. JMX enables external programs like the eG agent to connect to the JRE of an application and pull out metrics in real-time.

Note:

This section discusses the procedure for enabling JMX support for the JRE of any generic Java application that may be monitored using eG Enterprise. To know how to enable JMX support for the JRE of key application servers monitored out-of-the-box by eG Enterprise, refer to the relevant chapters of the Configuring and Monitoring Application Servers document.

By default, JMX requires no authentication or security (SSL). In this case therefore, to use JMX for pulling out metrics from a target application, the following will have to be done:

1. Login to the application host.
2. The **<JAVA_HOME>\jre\lib\management** folder used by the target application will typically contain the following files:
   - management.properties
   - jmxremote.access
   - jmxremote.password.template
   - snmp.acl.template
3. Edit the managerment.properties file, and append the following lines to it:

   ```
   com.sun.management.jmxremote.port=<Port No>
   ```

   ```
   com.sun.management.jmxremote.ssl=false com.sun.management.jmxremote.authenticate=false
   ```

   For instance, if the JMX listens on port 9005, then the first line of the above specification would be:

   ```
   com.sun.management.jmxremote.port=9005
   ```

4. Then, save the file.
5. Next, edit the start-up script of the target application, and add the following line to it:

   ```
   -Dcom.sun.management.config.file=<management.properties_file_path>
   ```

   ```
   -Djava.rmi.server.hostname=<IP Address>
   ```

6. For instance, on a Windows host, the <management.properties_file_path> can be expressed as: D:\bea\jrockit_150_11\jre\lib\management\management.properties
7. On other hand, on a Unix/Linux/Solaris host, a sample <management.properties_file_path> specification will be as follows: **/usr/jdk1.5.0_05/jre/lib/management/management.properties**

8. In the second line, set the <IP Address> to the IP address using which the Java application has been managed in the eG Enterprise system. Alternatively, you can add the following line to the startup script: -Djava.rmi.server.hostname=localhost
9. Save this script file too.
10. Next, during test configuration, do the following:
    - Set JMX as the mode;
    - Set the port that you defined in step 3 above (in the management.properties file) as the jmx remote port;
    - Set the user and password parameters to none.
    - Update the test configuration.

On the other hand, if JMX requires only authentication (and no security), then the following steps will apply:

1. Login to the application host. If the application is executing on a Windows host, then, login to the host as a local/domain administrator.
2. As stated earlier, the <java_home>\jre\lib\management folder used by the target application will typically contain the following files:
    - management.properties
    - jmxremote.access
    - jmxremote.password.template
    - snmp.acl.template
3. First, copy the jmxremote.password.template file to any other location on the host, rename it as as jmxremote.password, and then, copy it back to the **<JAVA_HOME>\jre\lib\management** folder.
4. Next, edit the jmxremote.password file and the jmxremote.access file to create a user with read-write access to the JMX. To know how to create such a user, refer to Section **2.1.2**.
5. Then, proceed to make the jmxremote.password file secure by granting a single user “full access” to that file. For monitoring applications executing on Windows in particular, only the Owner of the jmxremote.password file should have full control of that file. To know how to grant this privilege to the Owner of the file, refer to Section **2.1.1**.
6. In case of applications executing on Solaris / Linux hosts on the other hand, any user can be

granted full access to the jmxremote.password file, by following the steps below:

- Login to the host as the user who is to be granted full control of the jmxremote.password file.
- Issue the following command:

  chmod 600 jmxremote.password

- This will automatically grant the login user full access to the jmxremote.password file.

7. Next, edit the management.properties file, and append the following lines to it:

```
com.sun.management.jmxremote.port=<Port No>
com.sun.management.jmxremote.ssl=false
com.sun.management.jmxremote.authenticate=true
com.sun.management.jmxremote.access.file=<Path of jmxremote.access>
com.sun.management.jmxremote.password.file=<Path of jmxremote.password>
```

   For instance, assume that the JMX remote port is 9005, and the jmxremote.access and jmxremote.password files exist in the following directory on a Windows host: D:\bea\jrockit_150_11\jre\lib\management. The specification above will then read as follows:

```
com.sun.management.jmxremote.port=9005
com.sun.management.jmxremote.access.file=D:\\bea\\jrockit_150_11\\jre\\lib\\management\\jmxremote.access
com.sun.management.jmxremote.password.file=D:\\bea\\jrockit_150_11\\jre\\lib\\management\\jmxremote.password
```

8. If the application in question is executing on a Unix/Solaris/Linux host, and the jmxremote.access and jmxremote.password files reside in the **/usr/jdk1.5.0_05/jre/lib/management** folder of the host, then the last 2 lines of the specification will be:

```
com.sun.management.jmxremote.access.file=/usr/jdk1.5.0_05/jre/lib/management/jmxremote.access
com.sun.management.jmxremote.password.file=/usr/jdk1.5.0_05/jre/lib/management/jmxremote.password
```

9. Finally, save the file.

10. Then, edit the start-up script of the target web application server, include the following line in it, and save the file:

```
-Dcom.sun.management.config.file=<management.properties_file_path>
-Djava.rmi.server.hostname=<IP Address>
```

For instance, on a Windows host, the <management.properties_file_path> can be expressed as: **D:\bea\jrockit_150_11\jre\lib\management\management.properties.** On other hand, on a Linux/Solaris host, a sample<management.properties_file_path> specification will be as follows: **/usr/jdk1.5.0_05/jre/lib/management/management.properties**

11. In the second line, set the **<IP Address>** to the IP address using which the Java application has been managed in the eG Enterprise system. Alternatively, you can add the following line to the startup script of the target web application server: -Djava.rmi.server.hostname=localhost
12. Next, during test configuration, do the following:

    - Set **JMX** as the mode;
    - Ensure that the port number configured in the management.properties file at step 5 above is set as the jmx remote port;
    - Make sure that the user and password parameters of the test are that of a user with readwrite rights to JMX. To know how to create a new user and assign the required rights to him/her, refer to Section **2.1.2**.

    Note:

    eG Enterprise cannot use JMX that requires both authentication and security (SSL), for monitoring the target Java application.

2.1.1 Securing the 'jmxremote.password' file

To enable the eG agent to use JMX (that requires **authentication only**) for monitoring a Windows-based Java application, you need to ensure that the jmxremote.password file in the **<JAVA_HOME>\jre\lib\management** folder used by the target application is accessible **only by the Owner of that file**. To achieve this, do the following:

1. Login to the Windows host as a local/domain administrator.
2. Browse to the location of the jmxremote.password file using Windows Explorer.
3. Next, right-click on the jmxremote.password file and select the **Properties** option (see Figure 2.1).

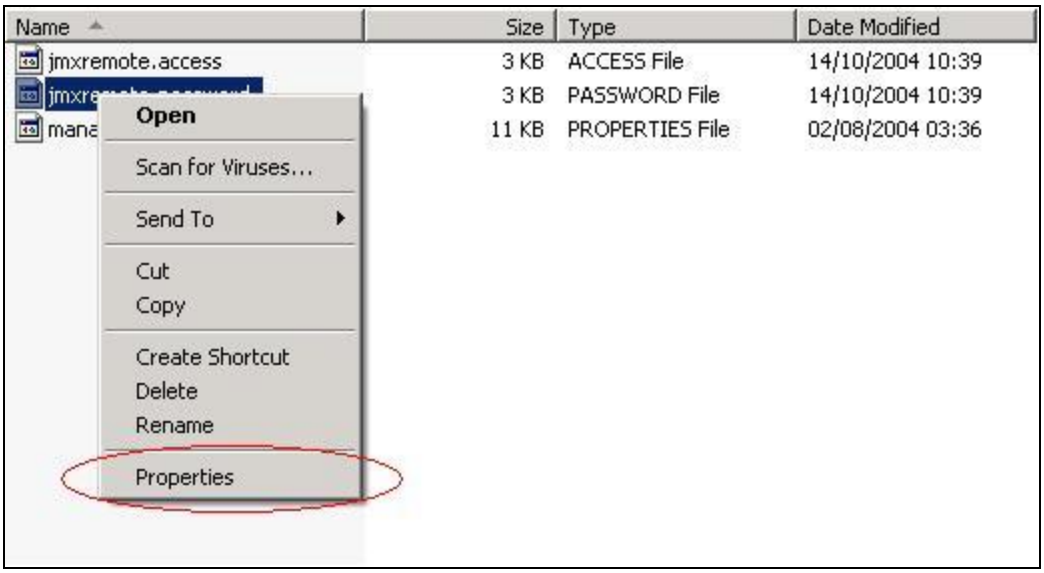


Figure 2.1: Selecting the Properties option

4. From Figure 2.2 that appears next, select the **Security** tab.

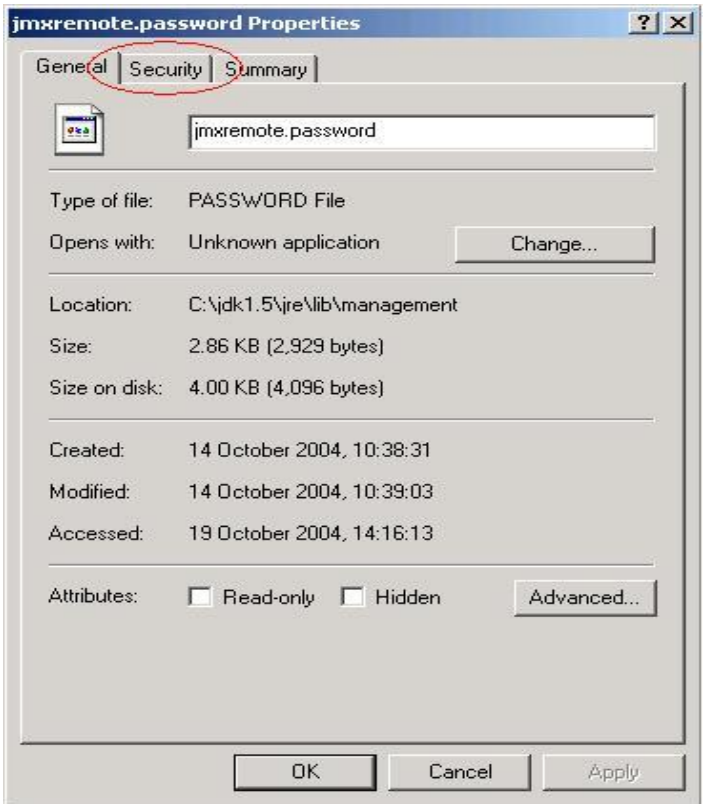


Figure 2.2: The Properties dialog box

However, if you are on Windows XP and the computer is not part of a domain, then the **Security** tab may be missing. To reveal the **Security** tab, do the following:

- Open Windows Explorer, and choose **Folder Options** from the **Tools** menu.
- Select the **View** tab, scroll to the bottom of the **Advanced Settings** section, and clear the check box next to **Use Simple File Sharing**.

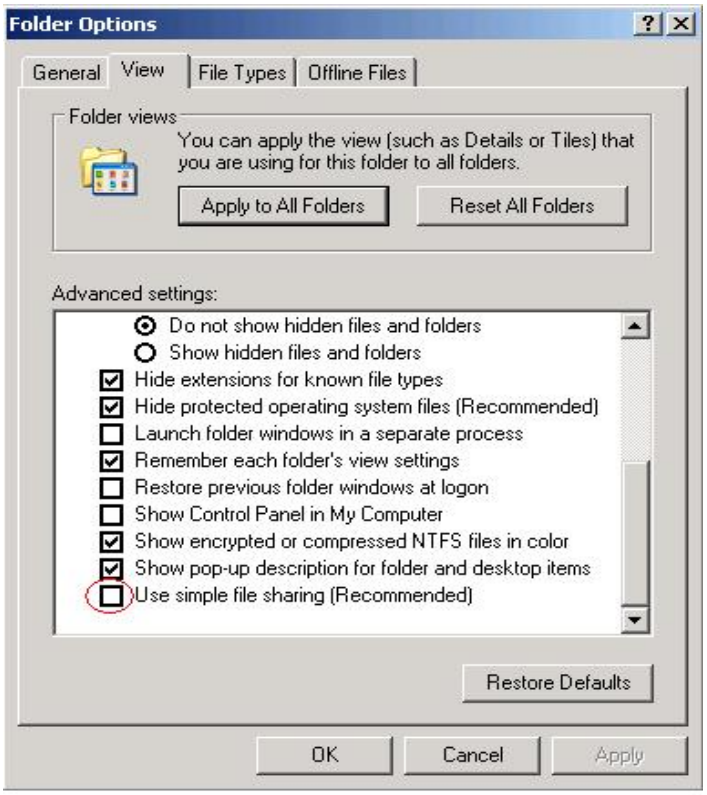


Figure 2.3: Deselecting the 'Use simple file sharing' option

- Click **OK** to apply the change
- When you restart Windows Explorer, the **Security** tab would be visible.

5. Next, select the **Advanced** button in the **Security** tab of Figure 2.4.

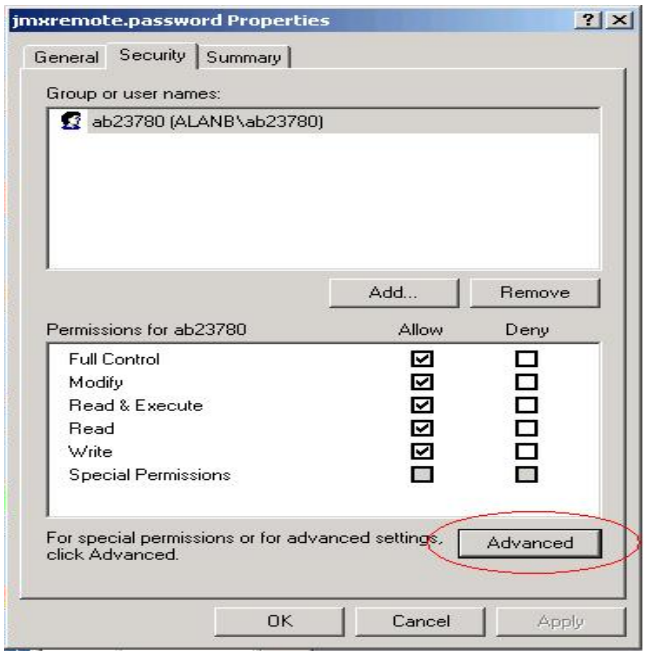


Figure 2.4: Clicking the Advanced button

6. Select the **Owner** tab to see who the owner of the file is.

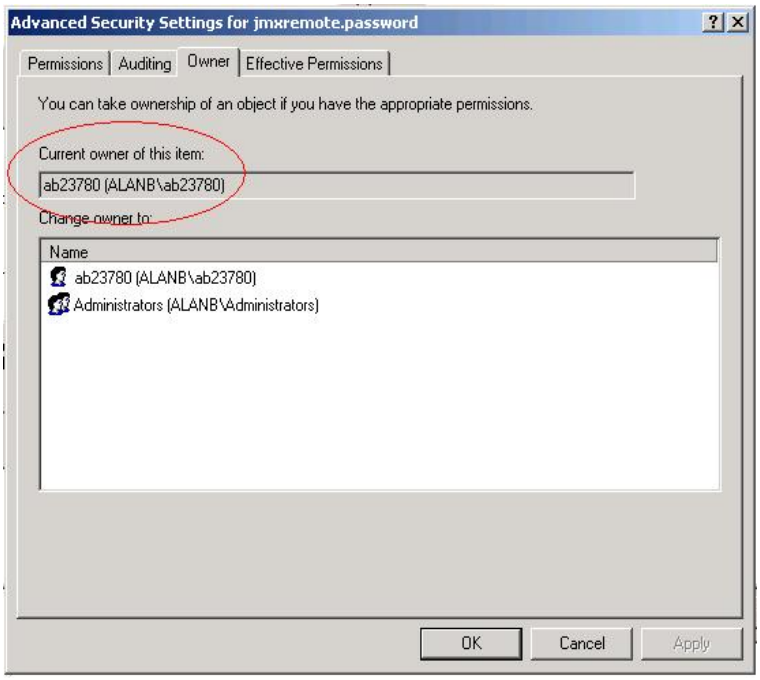


Figure 2.5: Verfying whether the Owner of the file is the same as the application Owner

7. Then, proceed to select the **Permissions** tab in Figure 2.5 to set the permissions. If the

jmxremote.password file has inherited its permissions from a parent directory that allows users or groups other than the **Owner** to access the file, then clear the **Inherit from parent the permission entries that apply to child objects** check box in Figure 2.6.

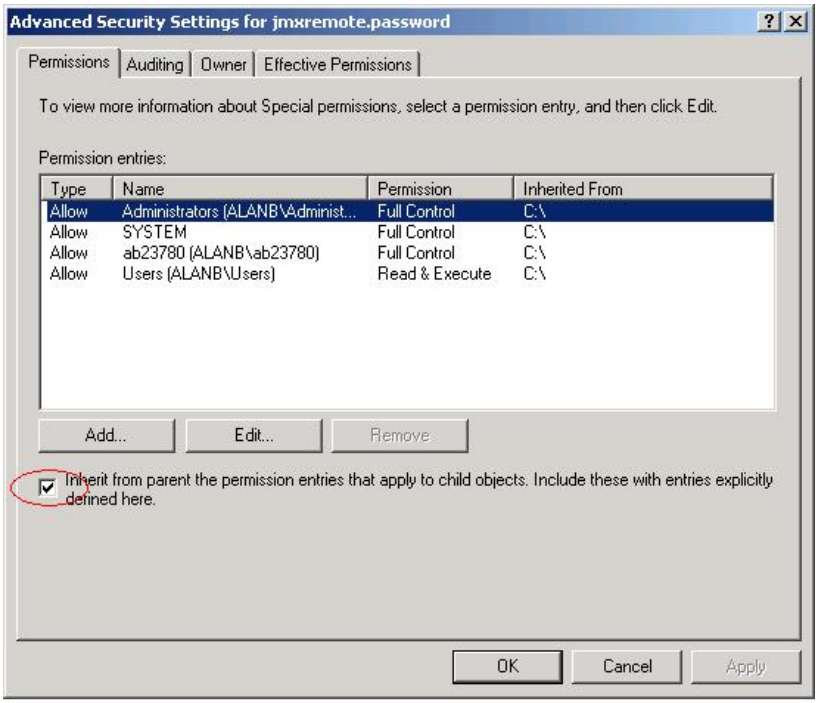


Figure 2.6: Disinheriting permissions borrowed from a parent directory

8. At this point, you will be prompted to confirm whether the inherited permissions should be copied from the parent or removed. Press the **Copy** button in Figure 2.7.

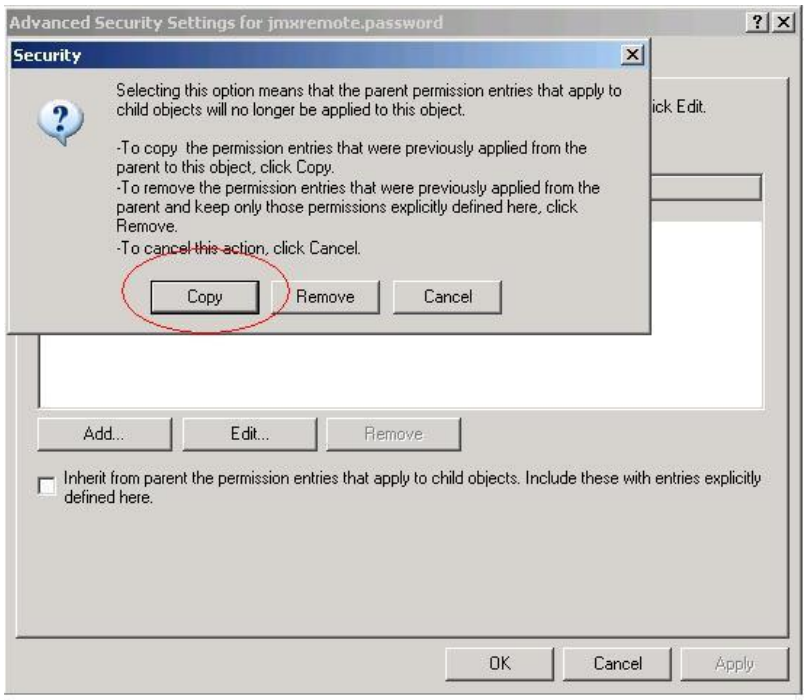


Figure 2.7: Copying the inherited permissions

9. Next, remove all permission entries that allow the jmxremote.password file to be accessed by users or groups other than the file **Owner**. For this, click the user or group and press the **Remove** button in Figure 2.8. At the end of this exercise, only a single permission entry granting **Full Control** to the owner should remain in Figure 2.8.

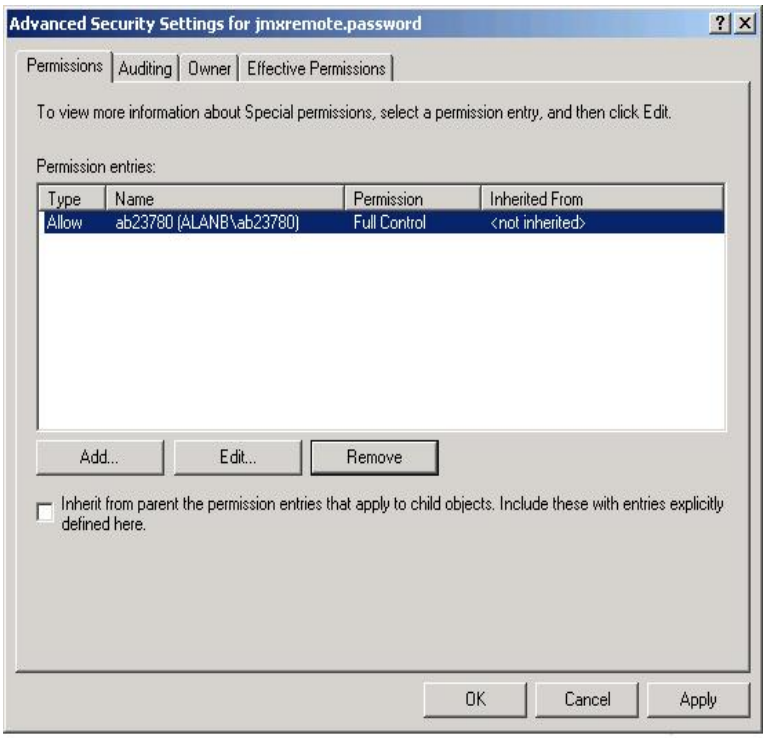


Figure 2.8: Granting full control to the file owner

10. Finally, click the **Apply** and **OK** buttons to register the changes. The password file is now secure, and can only be accessed by the file owner.

Note:

If you are trying to enable JMX on a Linux host, you might encounter issues with the way hostnames are resolved.

To solve it you might have to set the **-Djava.rmi.server.hostname=<hostname or localhost or ip>** property in the startup script of the target web application server.

If you are in local, simply try with **- Djava.rmi.server.hostname=localhost** or **- Djava.rmi.server.hostname=127.0.0.1**.

2.1.2 Configuring the eG Agent to Support JMX Authentication

If the eG agent needs to use JMX for monitoring a Java application, and this JMX requires **authentication only** (and not security), then every test to be executed by such an eG agent should be configured with the credentials of a valid user to JMX, with read-write rights. The steps for creating such a user are detailed below:

1. Login to the application host. If the application being monitored is on a Windows host, then login as a local/domain administrator to the host.
2. Go to the **<JAVA_HOME>\jre\lib\management** folder used by the target application to view the following files:
   - management.properties
   - jmxremote.access
   - jmxremote.password.template
   - snmp.acl.template
3. Copy the jmxremote.password.template file to a different location, rename it as jmxremote.password, and copy it back to the **<JAVA_HOME>\jre\lib\management** folder.
4. Open the jmxremote.password file and scroll down to the end of the file. By default, you will find the commented entries indicated by Figure 2.9 below:

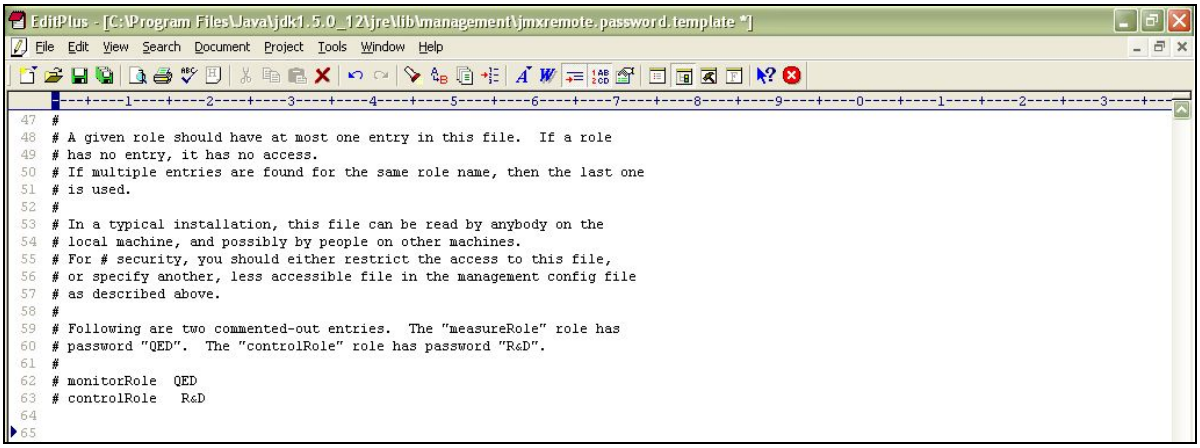


Figure 2.9: Scrolling down the jmxremote.password file to view 2 commented entries

5. The two entries indicated by Figure 2.9 are sample username password pairs with access to JMX. For instance, in the first sample entry of Figure 2.9 monitorRole is the username and QED is the password corresponding to monitorRole. Likewise, in the second line, the controlRole user takes the password R&D.
6. If you want to use one of these pre-defined username password pairs during test configuration, then simply uncomment the corresponding entry by removing the # symbol preceding that entry. However, prior to that, you need to determine what privileges have been granted to both these users. For that, open the jmxremote.access file in the editor.

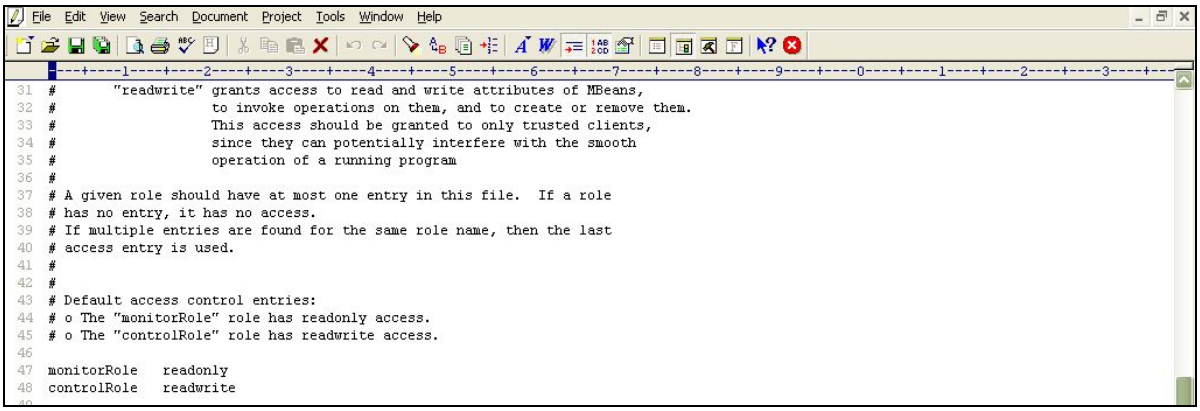


Figure 2.10: The jmxremote.access file

7. Scrolling down the file (as indicated by Figure 2.10) will reveal 2 lines, each corresponding to the sample username available in the jmxremote.password file. Each line denotes the access rights of the corresponding user. As is evident from Figure 2.10, the user monitorRole has only readonly rights, while user controlRole has readwrite rights. Since the eG agent requires readwrite rights to be able to pull out key JVM-related statistics using JMX, we will have to configure the test with the credentials of the user controlRole.
8. For that, first, edit the jmxremote.password file and uncomment the controlRole <password> line as depicted by Figure 2.11.

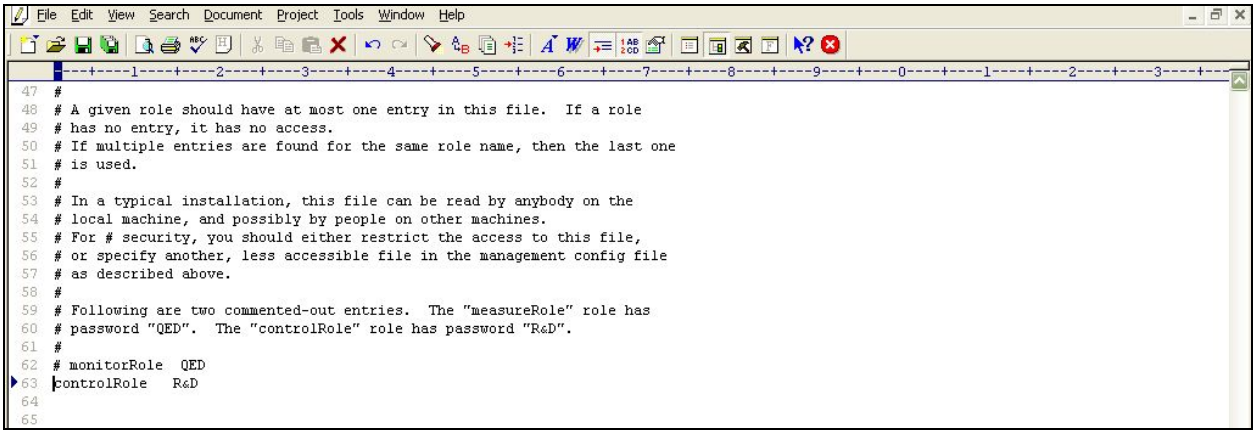


Figure 2.11: Uncommenting the 'controlRole' line

9. Then, save the file.
10. If a security manager is enabled for the target Java application, then you also need to make sure that the controlRole is allowed full access to the javax.management.MBeanPermission. To grant full access to the controlRole, do the following:

- Edit the java.policy file in the **<JAVA_HOME>\jre\lib\security** directory of the target Java application.
- Insert the following entry at the top of the file's contents (see Figure 2.12):

```
grant principal javax.management.remote.JMXPrincipal "controlRole" {
permission javax.management.MBeanPermission "*", "*";
};
```

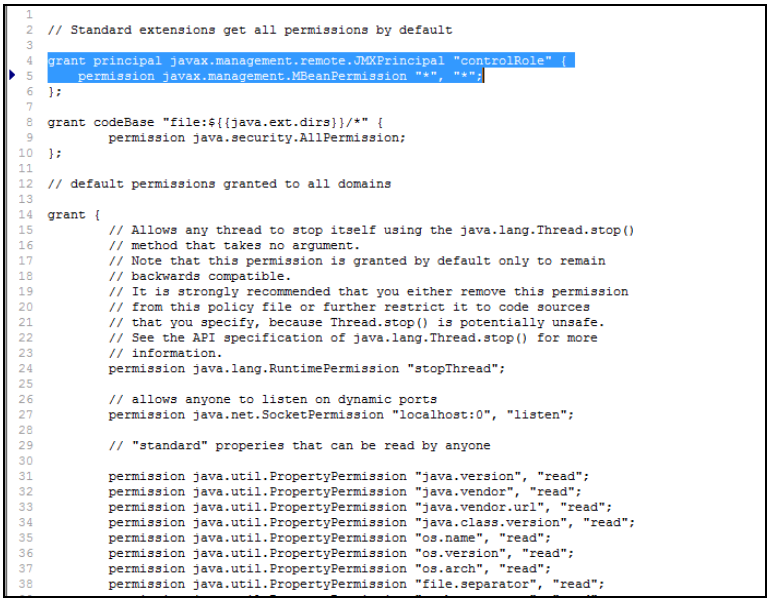


Figure 2.12: Editing the java.policy file

- Then, save the file.

  If this is not done, then a java.security.AccessControlException will occur, when the eG agent attempts to connect to the JRE using JMX.

11. You can now proceed to configure the tests with the user name controlRole and password R&D.
12. Alternatively, instead of going with these default credentials, you can create a new username password pair in the jmxremote.password file, assign readwrite rights to this user in the jmxremote.access file, and then configure the eG tests with the credentials of this new user. For instance, let us create a user john with password john and assign readwrite rights to john.
13. For this purpose, first, edit the jmxremote.password file, and append the following line (see Figure 2.13) to it:

john john

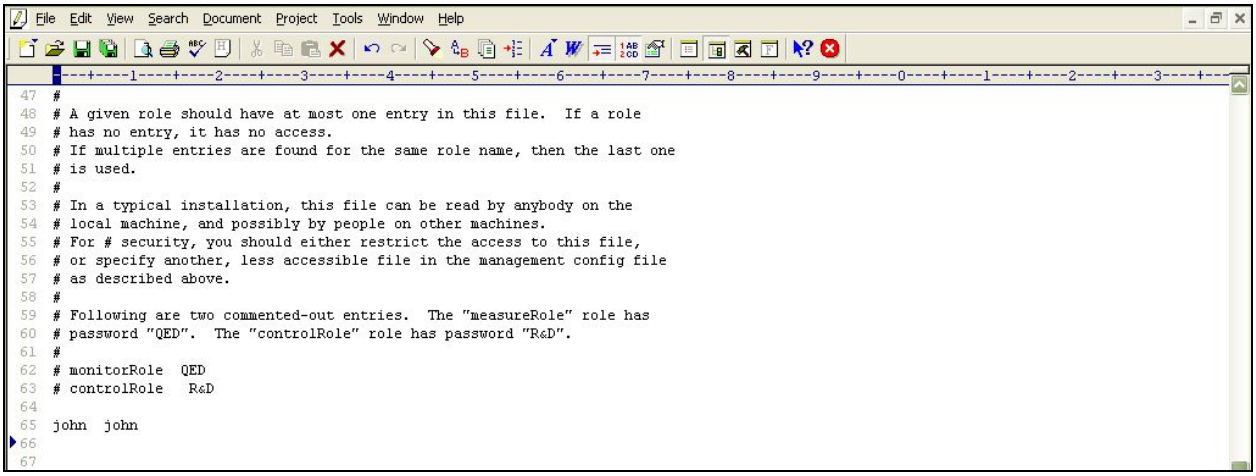


Figure 2.13: Appending a new username password pair

14. Save the jmxremote.password file.
15. Then, edit the jmxremote.access file, and append the following line (see Figure 2.14) to it:

    john readwrite

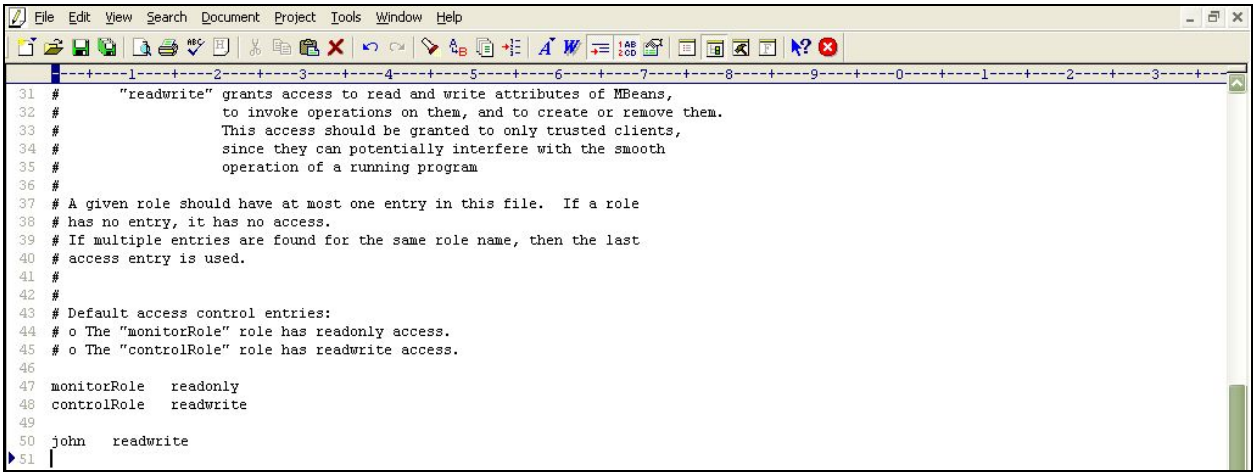
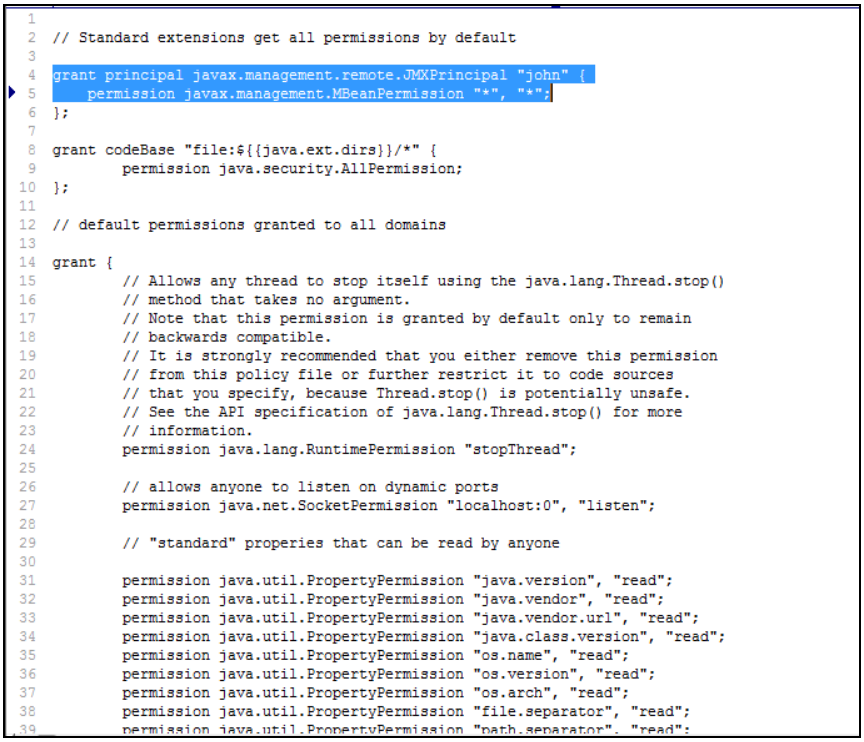


Figure 2.14: Assigning rights to the new user in the jmxremote.access file

16. Then, save the jmxremote.access file.
17. Next, if a security manager object is enabled for your Java application, then insert the following entry at the top of the contents of the java.policy file (in the <JAVA_HOME>\jre\lib\security folder (see Figure 2.15).



```
// Standard extensions get all permissions by default

grant principal javax.management.remote.JMXPrincipal "john" {
    permission javax.management.MBeanPermission "*", "*";
};

grant codeBase "file:${{java.ext.dirs}}/*" {
        permission java.security.AllPermission;
};

// default permissions granted to all domains

grant {
        // Allows any thread to stop itself using the java.lang.Thread.stop()
        // method that takes no argument.
        // Note that this permission is granted by default only to remain
        // backwards compatible.
        // It is strongly recommended that you either remove this permission
        // from this policy file or further restrict it to code sources
        // that you specify, because Thread.stop() is potentially unsafe.
        // See the API specification of java.lang.Thread.stop() for more
        // information.
        permission java.lang.RuntimePermission "stopThread";

        // allows anyone to listen on dynamic ports
        permission java.net.SocketPermission "localhost:0", "listen";

        // "standard" properies that can be read by anyone

        permission java.util.PropertyPermission "java.version", "read";
        permission java.util.PropertyPermission "java.vendor", "read";
        permission java.util.PropertyPermission "java.vendor.url", "read";
        permission java.util.PropertyPermission "java.class.version", "read";
        permission java.util.PropertyPermission "os.name", "read";
        permission java.util.PropertyPermission "os.version", "read";
        permission java.util.PropertyPermission "os.arch", "read";
        permission java.util.PropertyPermission "file.separator", "read";
        permission java.util.PropertyPermission "path.separator", "read";
```

Figure 2.15: Editing the java.policy file to grant john full access to javax.management.MBeanPermission

18. Finally, proceed to configure the tests with the user name and password, john and john, respectively.

2.2 Enabling SNMP Support for JRE

Instead of JMX, you can configure the eG agent to monitor a Java application using SNMP-based access to the Java runtime MIB statistics.

In some environments, SNMP access might have to be authenticated by an ACL (Access Control List), and in some other cases, it might not require an ACL.

If SNMP access **does not require ACL authentication**, then follow the steps below to enable SNMP support:

1. Login to the application host.
2. Ensure that the SNMP service and the SNMP Trap Service are running on the host.
3. Next, edit the management.properties file in the **<JAVA_HOME>\jre\lib\management** folder used by the target application.

4. Append the following lines to the file:

```
com.sun.management.snmp.port=<Port No>
com.sun.management.snmp.interface=0.0.0.0
com.sun.management.snmp.acl=false
```

For instance, if the SNMP port is 1166, then the first line of the above specification will be:

```
com.sun.management.snmp.port=1166
```

If the second line of the specification is set to 0.0.0.0, then, it indicates that the JRE will accept SNMP requests from any host in the environment. To ensure that the JRE services only those SNMP requests that are received from the eG agent, set the second line of the specification to the IP address of the agent host. For instance, if the eG agent to monitor the Java application is executing on 192.168.10.152, then the second line of the specification will be:

```
com.sun.management.snmp.interface=192.168.10.152
```

5. Next, edit the start-up script of the target application, include the following line it, and save the script file.

```
-Dcom.sun.management.config.file=<management.properties_file_path>
```

6. For instance, on a Windows host, the <management.properties_file_path> can be expressed as: **D:\bea\jrockit_150_11\jre\lib\management\management.properties**.
7. On other hand, on a Unix/Linux/Solaris host, a sample <management.properties_file_path> specification will be as follows: **/usr/jdk1.5.0_05/jre/lib/management/management.properties**.

On the contrary, if SNMP access requires **ACL authentication**, then follow the steps below to enable SNMP support for the JRE:

1. Login to the application host. If the target application is executing on a Windows host, login as a local/domain administrator.
2. Ensure that the SNMP service and SNMP Trap Service are running on the host.
3. Copy the snmp.acl.template file in the **<JAVA_HOME>\jre\lib\management** folder to another location on the local host. Rename the snmap.acl.template file as snmp.acl, and copy the snmp.acl file back to the **<JAVA_HOME>\jre\lib\management** folder.
4. Next, edit the snmp.acl file, and set rules for SNMP access in the file.

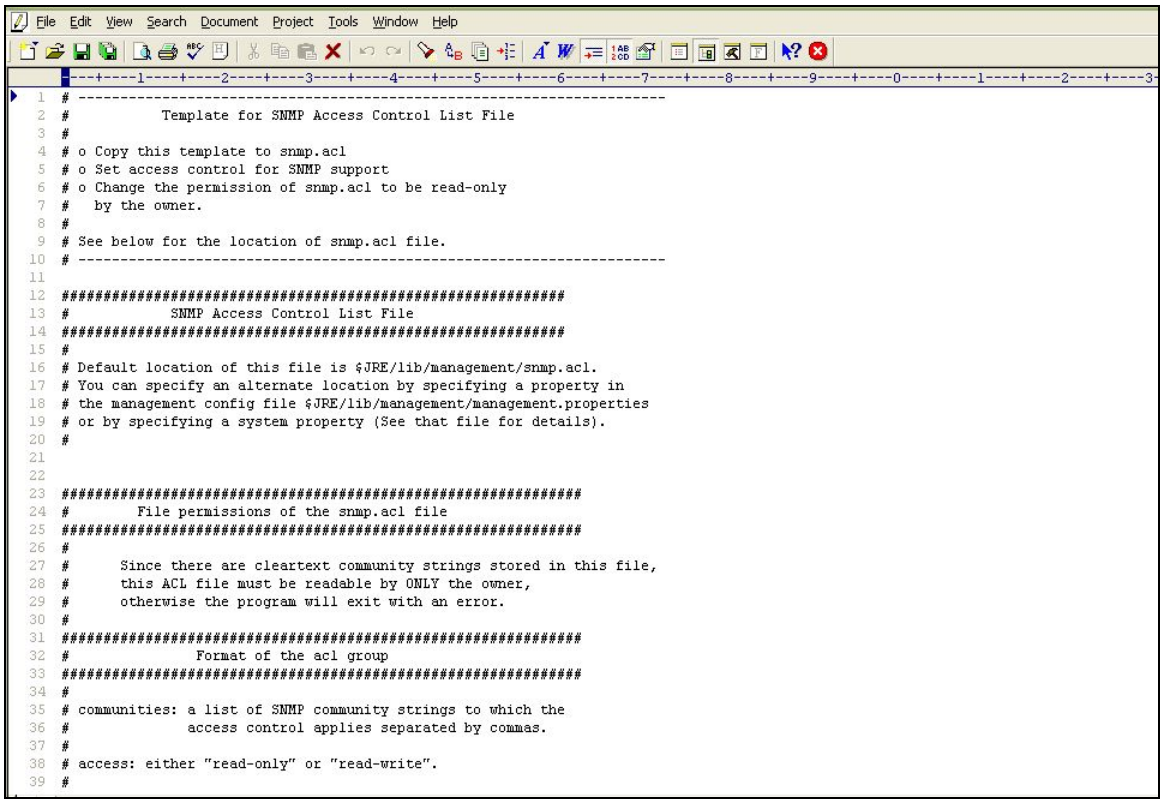


Figure 2.16: The snmp.acl file

5. For that, first scroll down the file to view the sample code block revealed by Figure 2.17.

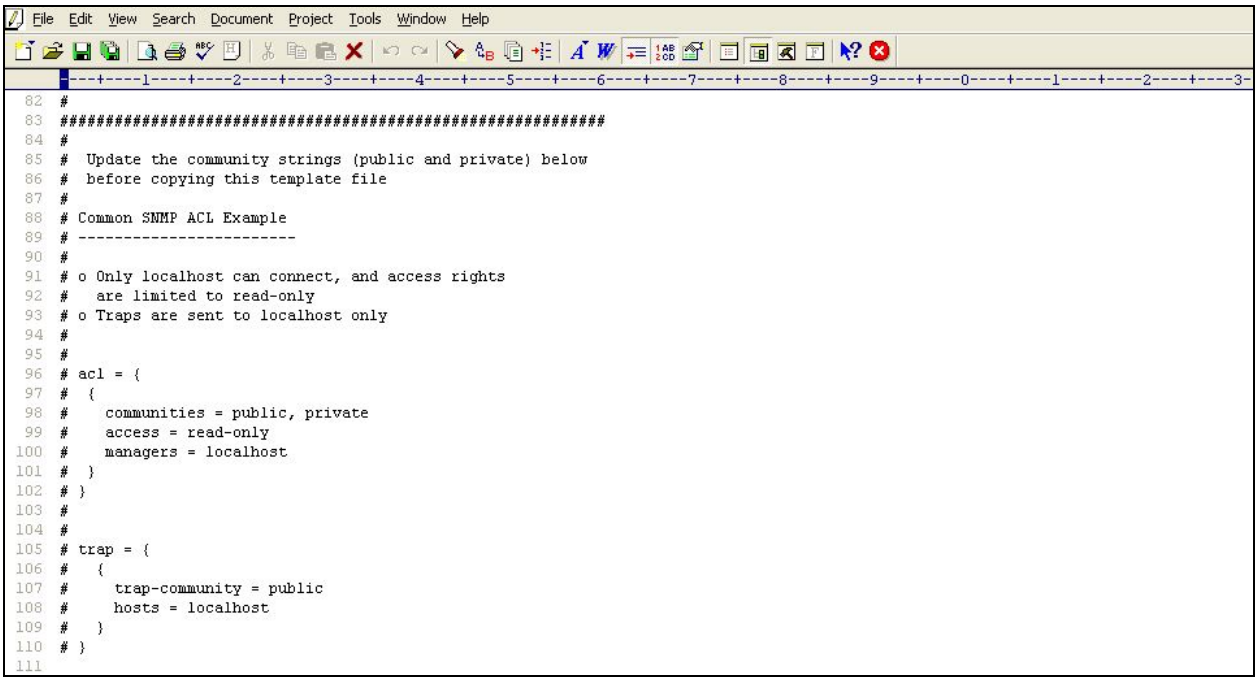


Figure 2.17: The snmp.acl file revealing the SNMP ACL example

6. Uncomment the code block by removing the # symbol preceding each line of the block as indicated by Figure 2.18.

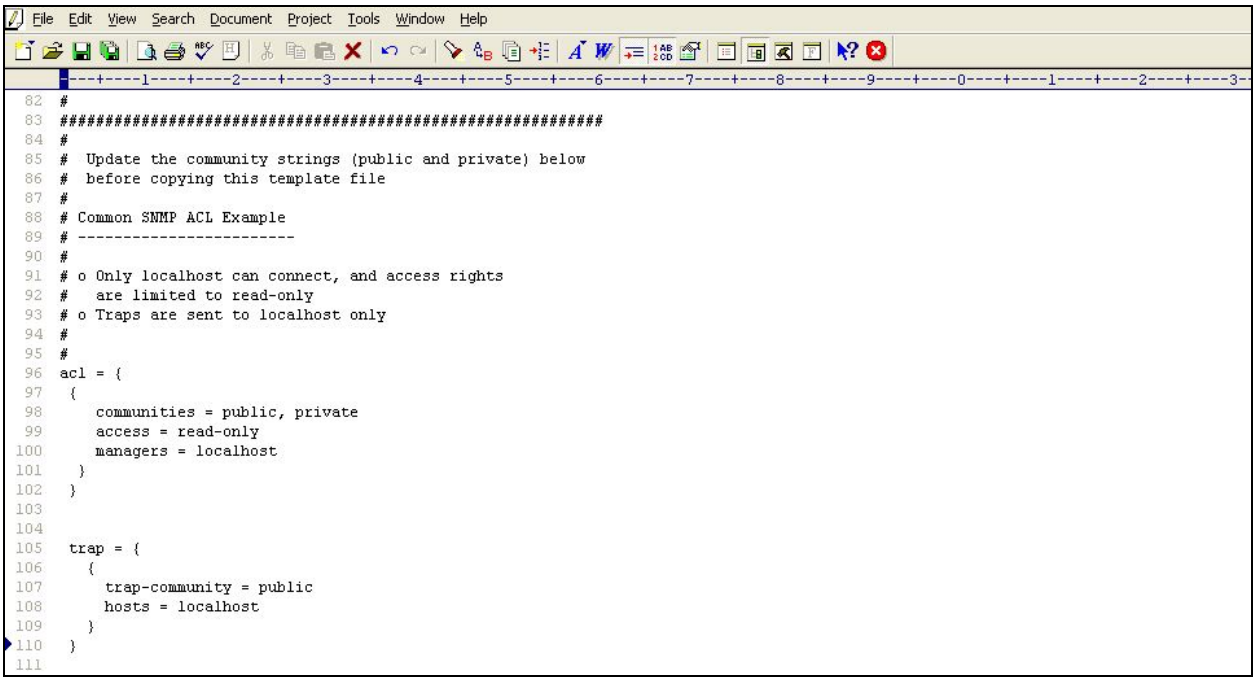


Figure 2.18: Uncommenting the code block

7. Next, edit the code block to suit your environment.
8. The acl block expects the following parameters:

   - communities : Provide a comma-separated list of community strings, which an SNMP request should carry for it to be serviced by this JRE; in the example illustrated by Figure 17, the community strings recognized by this JRE are public and private. You can add more to this list, or remove a community string from this list, if need be.
   - access : Indicate the access rights that SNMP requests containing the defined communities will have; in Figure 2.18, SNMP requests containing the community string public or private, will have only read-only access to the MIB statistics. To grant full access, you can specify rea-write instead.
   - managers : Specify a comma-separated list of SNMP managers or hosts from which SNMP requests will be accepted by this JRE; in the example illustrated by Figure 2.18, all SNMP requests from the localhost will be serviced by this JRE. Typically, since the SNMP requests originate from an eG agent, the IP of the eG agent should be configured against the managers parameter. For instance, if the IP address of the agent host is 192.16.10.160, then, to ensure that the JRE accepts requests from the eG agent alone, set managers to 192.168.10.160, instead of localhost.

9. Every acl block in the snmp.acl file should have a corresponding trap block. This trap block should be configured with the following values:

   - trap-community: Provide a comma-separated list of community strings that can be used by SNMP traps sent by the Java application to the managers specified in the acl block. In the example of Figure 2.18, all SNMP traps sent by the Java application being monitored should use the community string public only.
   - hosts: Specify a comma-separated list of IP addresses / host names of hosts from which SNMP traps can be sent. In the case of Figure 2.18, traps can be sent by the localhost only. If a single snmp.acl file is being centrally used by multiple applications/devices executing on multiple hosts, then to ensure that all such applications are able to send traps to the configured SNMP managers (in the acl block), you can provide the IP address/hostname of these applications as a comma-separated list against hosts.

10. Figure 2.19 depicts how the *acl* and *trap* blocks can be slightly changed to suit the monitoring needs of an application.

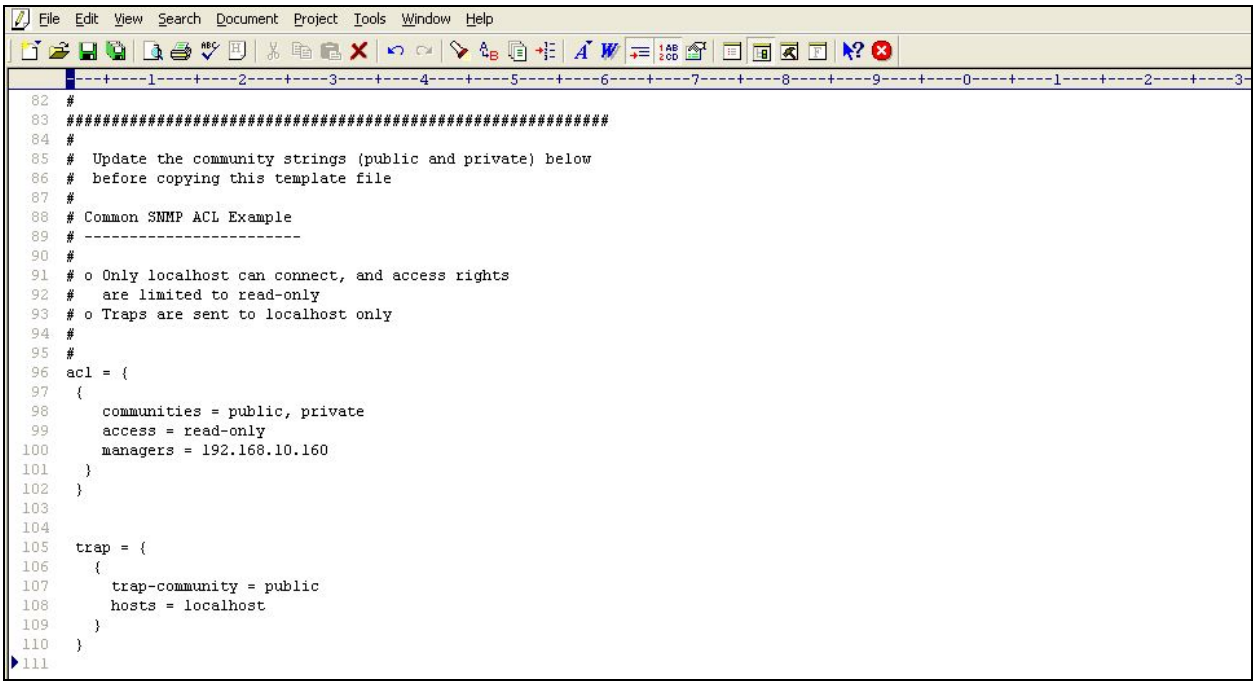


Figure 2.19: The edited block

11. Then, proceed to make the snmp.acl file secure by granting a single user “full access” to that file. For monitoring applications executing on Windows in particular, only the Owner of the snmp,.acl file should have full control of that file. To know how to grant this privilege to the Owner of a file, refer to Section **2.1.1** . This section actually details the procedure for making the jmxremote.password file on Windows, secure. Use the same procedure for making the snmp.acl file on Windows secure, but make sure that you select the snmp.acl file and not the jmxremote.password file.
12. In case of applications executing on Solaris / Linux hosts on the other hand, any user can be granted full access to the snmp.acl file, by following the steps below:
    - Login to the host as the user who is to be granted full control of the snmp.acl file.
    - Issue the following command:

      chmod 600 snmp.acl

    - This will automatically grant the login user full access to the jmxremote.password file.
13. Next, edit the *management.properties* file in the <JAVA_HOME>\jre\lib\management folder used by the target application.
14. Append the following lines to the file:

```
com.sun.management.snmp.port=<PortNo>
com.sun.management.snmp.interface=0.0.0.0
com.sun.management.snmp.acl=true
com.sun.management.snmp.acl.file=<Path_of_snmp.acl>
```

If the second line of the specification is set to *0.0.0.0*, then, it indicates that the JRE will accept SNMP requests from any host in the environment. To ensure that the JRE services only those SNMP requests that are received from the eG agent, set the second line of the specification to the IP address of the agent host.

For example, if the Java application being monitored listens for SNMP requests at port number 1166, the eG agent monitoring the Java application is deployed on 192.168.10.152, and these SNMP requests need to be authenticated using the snmp.acl file in the **D:\bea\jrockit_150_11\jre\lib** directory, then the above specification will read as follows:

```
com.sun.management.snmp.port=1166
com.sun.management.snmp.interface=192.168.10.152
com.sun.management.snmp.acl=true    com.sun.management.snmp.acl.file=D:\\bea\\jrockit_150_11\\jre\\lib\\management\\snmp.acl
```

15. However, if the application in question is executing on a Unix/Solaris/Linux host, and the snmp.acl file is in the **/usr/jdk1.5.0_05/jre/lib/management** folder of the host, then the last line of the specification will be:

```
com.sun.management.snmp.acl.file =/usr/jdk1.5.0_05/jre/lib/management/snmp.acl
```

16. Next, edit the start-up script of the target application, include the following line in it, and save the script file.

```
-Dcom.sun.management.config.file=<management.properties_file_path>
```

17. For instance, on a Windows host, the <management.properties_file_path> can be expressed as: **D:\bea\jrockit_150_11\jre\lib\management\management.properties**.
18. On other hand, on a Unix/Linux/Solaris host, a sample <management.properties_file_path> specification will be as follows: **/usr/jdk1.5.0_05/jre/lib/management/management.properties**.

The sections to come discuss the top 2 layers of Figure 3.1, as the remaining layers have already been discussed at length in the *Monitoring Unix and Windows Servers* document.

2.3 Managing the Java Application

The eG Enterprise cannot automatically discover a Java Application. This implies that you need to manually add the component for monitoring. To manage a Java Application component, do the following:

1. Log into the eG administrative interface.
2. eG Enterprise cannot automatically discover Java Application server. You need to manually add the server using the **COMPONENTS** page (see Figure 2.20) that appears when the Infrastructure -> Components -> Add/Modify menu sequence is followed. Remember that components manually added are managed automatically.

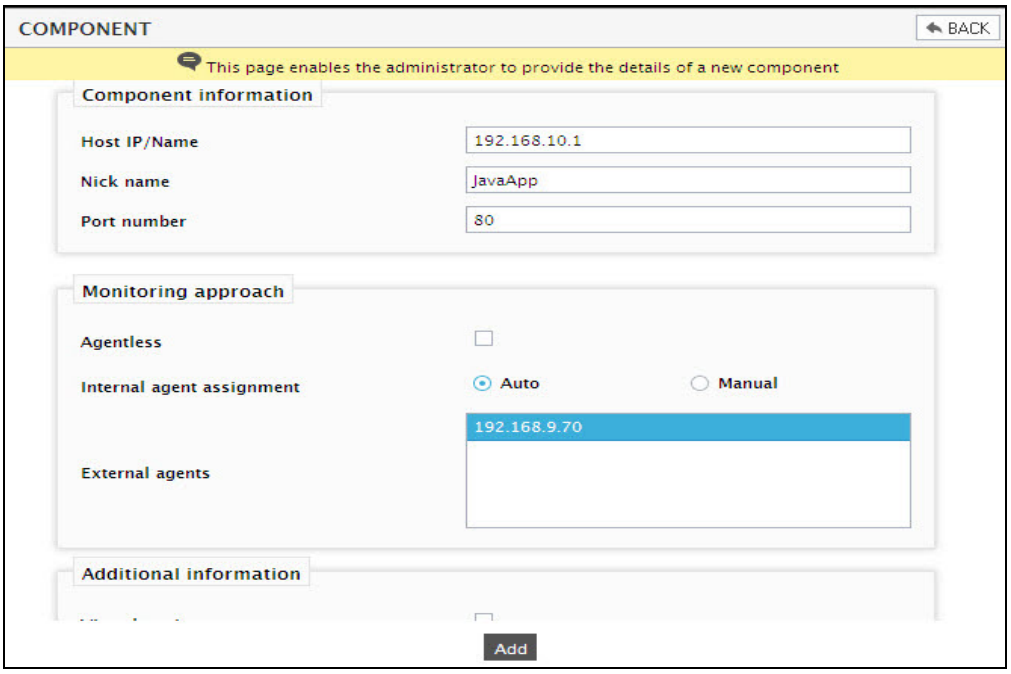
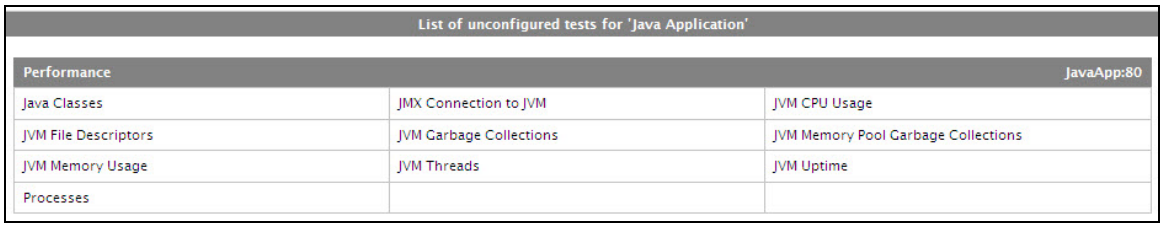


Figure 2.20: Adding a Java Application

3. When you attempt to sign out, a list of unconfigured tests appears.


List of unconfigured tests for 'Java Application'

Performance		JavaApp:80
Java Classes	JMX Connection to JVM	JVM CPU Usage
JVM File Descriptors	JVM Garbage Collections	JVM Memory Pool Garbage Collections
JVM Memory Usage	JVM Threads	JVM Uptime
Processes		

Figure 2.21: List of Unconfigured tests for the Java Application

4. Click on the **Java Classes** test to configure it. This test reports the number of classes loaded/unloaded from the memory. To know how to configure the test, refer to Section **3.2.3**.
5. Finally, signout of the eG administrative interface.

Chapter 3: Monitoring a Java Application

The prime concern of administrators of Java applications is knowing how well the application is functioning, and how to troubleshoot issues (if any) in the performance of these applications. Most web application server vendors prescribe custom APIs for monitoring – for instance, WebSphere and WebLogic allow administrators to use their built-in APIs for performance monitoring and problem detection.

Besides such applications, you might have stand-alone Java applications that do not provide any APIs for monitoring. To enable users to monitor the overall health of such stand-alone Java applications, eG Enterprise offers a generic monitoring model called the *Java Application*.

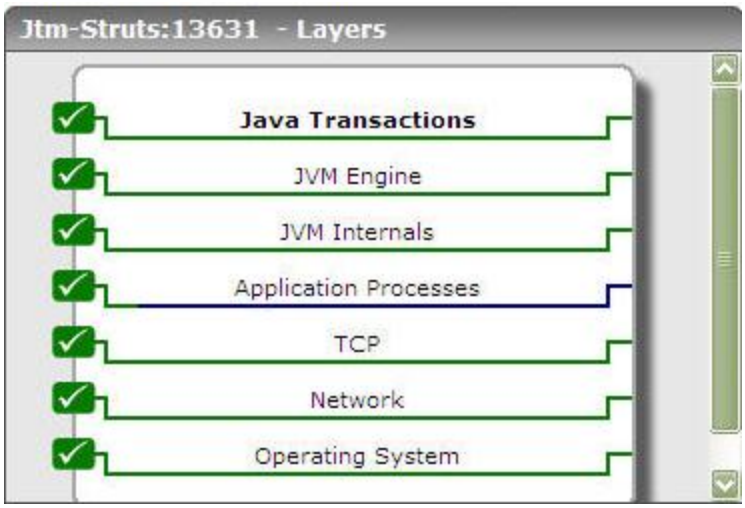


Figure 3.1: Layer model of the Java Application

Each layer of Figure 3.1 above is mapped to a series of tests that report critical statistics pertaining to the Java application being monitored. Using these statistics, administrators can figure out the following:

- Has the Java heap been sized properly?
- How effective is garbage collection? Is it impacting application performance?
- Often, Java programs use threads. A single program may involve multiple concurrent threads running in parallel. Is there excessive blocking between threads due to synchronization issues during application design?

- Are there runaway threads, which are taking too many CPU cycles? If such threads exist, which portions of code are responsible for spawning such threads?
- Is the JVM managing its memory resources efficiently or is the free memory on the JVM very less? Which type of memory is being utilized by the JVM increasingly?
- Has a scheduled JVM restart occurred? If so, when?

3.1 The Java Transactions Layer

By default, this layer will not be available for any monitored Java Application. This is because, the **Java Business Transactions** test mapped to this layer is disabled by default. To enable the test, follow the *Agents -> Tests -> Enable/Disable* menu sequence, select *Java Application* as the **Component type**, *Performance* as the **Test type**, and then select **Java Business Transactions** from the **DISABLED TESTS** list. Click the **Enable** button to enable the selected test, and click the **Update** button to save the changes.

3.1.1 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.

For this test to run and report metrics, you first need to **install and configure the eG Java BTM (Business Transaction Monitor) on the target Java application / J2EE container**. To know how, refer to the *Java Business Transaction Monitoring* document.

After BTM-enabling the target, configure this test. **Instructions for configuring this test and interpreting the metrics it reports are available in the 'Java Business Transactions Test' topic in the *Java Business Transaction Monitoring* document.**

3.2 The JVM Internals Layer

The tests associated with this layer measure the internal health of the Java Virtual Machine (JVM), and enables administrators to find accurate answers to the following performance queries:

- How many classes have been loaded/unloaded from memory?
- Did garbage collection take too long to complete? If so, which memory pools spent too much time in garbage collection?
- Are too many threads in waiting state in the JVM?
- Which threads are consuming CPU?

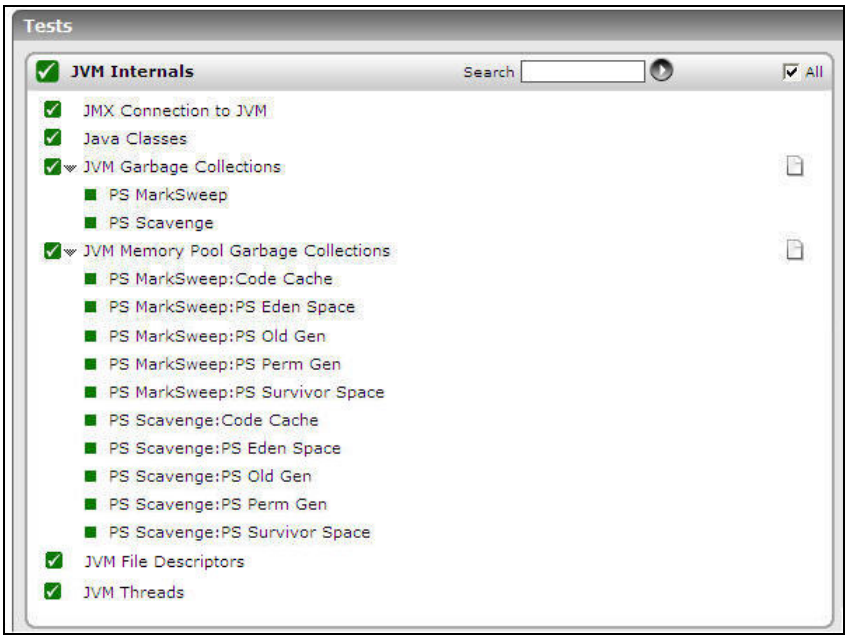


Figure 3.2: The tests associated with the JVM Internals layer

3.2.1 JMX Connection to JVM

This test reports the availability of the target Java application, and also indicates whether JMX is enabled on the application or not. In addition, the test promptly alerts you to slowdowns experienced by the application, and also reveals whether the application was recently restarted or not.

Target of the test : A Java application

Agent deploying the test : An internal/remote agent

Outputs of the test : One set of results for the Java application being monitored

Configurable parameters for the test

Parameter	Description
Test period	How often should the test be executed.
Host	The host for which the test is to be configured.
Port	The port number at which the specified host listens to.
JMX remote port	This parameter appers only if the mode is set to jmx. Here, specify the port at which the jmx listens for requests from remote hosts. Ensure that you specify the same port that you configured in the management.properties file in the

Parameter	Description
	<JAVA_HOME>\jre\lib\management folder used by the target application (see page **3**).
JNDIname	This parameter appears only if the mode is set to JMX. The JNDIname is a lookup name for connecting to the JMX connector. By default, this is jmxrmi. If you have resgistered the JMX connector in the RMI registery using a different lookup name, then you can change this default value to reflect the same.
User, Password, and Confirm password	These parameters appear only if the Mode is set to JMX. If JMX requires authentication only (but no security), then ensure that the user and password parameters are configured with the credentials of a user with read-write access to JMX. To know how to create this user, refer to Section **2.1.2**. Confirm the password by retyping it in the confirm password text box.
Provider	This parameter appears only if the Mode is set to JMX. This test uses a JMX Provider to access the MBean attributes of the target Java application and collect metrics. Specify the package name of this JMX Provider here. By default, this is set to com.sun.jmx.remote.protocol.
Timeout	Specify the duration (in seconds) for which this test should wait for a response from the target Java application. If there is no response from the target beyond the configured duration, the test will timeout. By default, this is set to 240 seconds if the Mode is JMX, and 10 seconds if the Mode is SNMP.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
JMX availability	Indicates whether the target application is available or not and whether JMX is enabled or not on the application.	Percent	If the value of this measure is 100%, it indicates that the Java application is available with JMX enabled. The value 0 on the other hand, could indicate one/both the following: • The Java application is unavailable; • The Java application is available, but JMX is not enabled;

Measurement	Description	Measurement Unit	Interpretation
JMX response time	Indicates the time taken to connect to the JMX agent of the Java application.	Secs	A high value could indicate a connection bottleneck.
Has the PID changed?	Indicates whether/not the process ID that corresponds to the Java application has changed.		This measure will report the value Yes if the PID of the target application has changed; such a change is indicative of an application restart. If the application has not restarted - i.e., if the PID has not changed - then this measure will return the value No.

3.2.2 JVM File Descriptors Test

This test reports useful statistics pertaining to file descriptors.

Note:

This test will work only if the target Java application uses the JDK/JRE offered by one of the following vendors only: Oracle, Sun, OpenJDK. IBM JDK/JRE is not supported.

Target of the test : A Java application

Agent deploying the test : An internal/remote agent

Outputs of the test : One set of results for the Java application being monitored

Configurable parameters for the test

Parameter	Description
Test period	How often should the test be executed.
Host	The host for which the test is to be configured.
Port	The port number at which the specified host listens to.
JMX remote port	This parameter appers only if the mode is set to jmx. Here, specify the port at which the jmx listens for requests from remote hosts. Ensure that you specify

Parameter	Description
	the same port that you configured in the management.properties file in the <JAVA_HOME>\jre\lib\management folder used by the target application (see page **3**).
JNDIname	This parameter appears only if the mode is set to JMX. The JNDIname is a lookup name for connecting to the JMX connector. By default, this is jmxrmi. If you have resgistered the JMX connector in the RMI registery using a different lookup name, then you can change this default value to reflect the same.
User, Password, and Confirm password	These parameters appear only if the Mode is set to JMX. If JMX requires authentication only (but no security), then ensure that the user and password parameters are configured with the credentials of a user with read-write access to JMX. To know how to create this user, refer to Section **2.1.2**. Confirm the password by retyping it in the confirm password text box.
Provider	This parameter appears only if the Mode is set to JMX. This test uses a JMX Provider to access the MBean attributes of the target Java application and collect metrics. Specify the package name of this JMX Provider here. By default, this is set to com.sun.jmx.remote.protocol.
Timeout	Specify the duration (in seconds) for which this test should wait for a response from the target Java application. If there is no response from the target beyond the configured duration, the test will timeout. By default, this is set to 240 seconds if the Mode is JMX, and 10 seconds if the Mode is SNMP.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Open file descriptors in JVM	Indicates the number of file descriptors currently open for the application.	Number	
Maximum file descriptors in JVM	Indicates the maximum number of file descriptors allowed for the application.	Number	
File descriptor usage by JVM	Indicates the file descriptor usage in percentage.	Percent	

3.2.3 Java Classes Test

This test reports the number of classes loaded/unloaded from the memory.

Target of the test : A Java application

Agent deploying the test : An internal/remote agent

Outputs of the test : One set of results for the Java application being monitored

Configurable parameters for the test

Parameter	Description
Test period	How often should the test be executed
Host	The host for which the test is to be configured.
Port	The port number at which the specified host listens to
Mode	This test can extract metrics from the Java application using either of the following mechanisms: • Using SNMP-based access to the Java runtime MIB statistics; • By contacting the Java runtime (JRE) of the application via JMX To configure the test to use SNMP, select the SNMP option. On the other hand, choose the JMX option to configure the test to use JMX instead. By default, the JMX option is chosen here.
JMX remote port	This parameter appers only if the mode is set to jmx. Here, specify the port at which the jmx listens for requests from remote hosts. Ensure that you specify the same port that you configured in the management.properties file in the <JAVA_HOME>\jre\lib\management folder used by the target application (see page **3**).
JNDIname	This parameter appears only if the mode is set to JMX. The JNDIname is a lookup name for connecting to the JMX connector. By default, this is jmxrmi. If you have resgistered the JMX connector in the RMI registery using a different lookup name, then you can change this default value to reflect the same.
User, Password, and Confirm	These parameters appear only if the Mode is set to JMX. If JMX requires authentication only (but no security), then ensure that the user and password

Parameter	Description
password	parameters are configured with the credentials of a user with read-write access to JMX. To know how to create this user, refer to Section **2.1.2**. Confirm the password by retyping it in the confirm password text box.
Provider	This parameter appears only if the Mode is set to JMX. This test uses a JMX Provider to access the MBean attributes of the target Java application and collect metrics. Specify the package name of this JMX Provider here. By default, this is set to com.sun.jmx.remote.protocol.
Timeout	Specify the duration (in seconds) for which this test should wait for a response from the target Java application. If there is no response from the target beyond the configured duration, the test will timeout. By default, this is set to 240 seconds if the Mode is JMX, and 10 seconds if the Mode is SNMP.
SNMPPort	This parameter appears only if the Mode is set to SNMP. Here specify the port number through which the server exposes its SNMP MIB. Ensure that you specify the same port you configured in the management.properties file in the <JAVA_HOME>\jre\lib\management folder used by the target application (see page **18**).
SNMP Version	This parameter appears only if the Mode is set to SNMP. The default selection in the SNMP version list is v1. However, for this test to work, you have to select SNMP v2 or v3 from this list, depending upon which version of SNMP is in use in the target environment.
SNMP Community	This parameter appears only if the Mode is set to SNMP. Here, specify the SNMP community name that the test uses to communicate with the mail server. The default is public. This parameter is specific to SNMP v1 and v2 only. Therefore, if the SNMP version chosen is v3, then this parameter will not appear.
User name	This parameter appears only when v3 is selected as the SNMP version. SNMP version 3 (SNMPv3) is an extensible SNMP Framework which supplements the SNMPv2 Framework, by additionally supporting message security, access control, and remote SNMP configuration capabilities. To extract performance statistics from the MIB using the highly secure SNMP v3 protocol, the eG agent has to be configured with the required access privileges – in other words, the eG agent should connect to the MIB using the credentials of a user with access permissions to be MIB. Therefore, specify the name of such a user against this parameter.

Parameter	Description
Context	This parameter appears only when v3 is selected as the SNMPVERSION. An SNMP context is a collection of management information accessible by an SNMP entity. An item of management information may exist in more than one context and an SNMP entity potentially has access to many contexts. A context is identified by the SNMPEngineID value of the entity hosting the management information (also called a contextEngineID) and a context name that identifies the specific context (also called a contextName). If the USERNAME provided is associated with a context name, then the eG agent will be able to poll the MIB and collect metrics only if it is configured with the context name as well. In such cases therefore, specify the context name of the username in the context text box. By default, this parameter is set to none.
Authpass	Specify the password that corresponds to the above-mentioned user name. This parameter once again appears only if the snmpversion selected is v3.
Confirm password	Confirm the Authpass by retyping it here
Authtype	This parameter too appears only if v3 is selected as the snmpversion. From the authtype list box, choose the authentication algorithm using which SNMP v3 converts the specified username and password into a 32-bit format to ensure security of SNMP transactions. You can choose between the following options: • MD5 – Message Digest Algorithm • SHA – Secure Hash Algorithm
Encryptflag	This flag appears only when v3 is selected as the snmpversion. By default, the eG agent does not encrypt SNMP requests. Accordingly, the flag is set to No by default. To ensure that SNMP requests sent by the eG agent are encrypted, select the Yes option.
Encrypttype	If the Encryptflag is set to Yes, then you will have to mention the encryption type by selecting an option from the Encrypttype list. SNMP v3 supports the following encryption types: • DES – Data Encryption Standard • AES – Advanced Encryption Standard
Encryptpassword	Specify the encryption password here.

Parameter	Description
Confirm password	Confirm the encryption password by retyping it here.
Data over TCP	This parameter is applicable only if mode is set to SNMP. By default, in an IT environment, all data transmission occurs over UDP. Some environments however, may be specifically configured to offload a fraction of the data traffic – for instance, certain types of data traffic or traffic pertaining to specific components – to other protocols like TCP, so as to prevent UDP overloads. In such environments, you can instruct the eG agent to conduct the SNMP data traffic related to the monitored target over TCP (and not UDP). For this, set this flag to Yes. By default, this flag is set to No.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Classes loaded	Indicates the number of classes currently loaded into memory.	Number	Classes are fundamental to the design of Java programming language. Typically, Java applications install a variety of class loaders (that is, classes that implement java.lang.ClassLoader) to allow different portions of the container, and the applications running on the container, to have access to different repositories of available classes and resources. A consistent decrease in the number of classes loaded and unloaded could indicate a road-block in the loading/unloading of classes by the class loader. If left unchecked, critical resources/classes could be rendered inaccessible to the application, thereby severely affecting its performance.
Classes unloaded	Indicates the number of classes currently unloaded from memory.	Number	
Total classes loaded	Indicates the total number of classes loaded into memory since the JVM started, including those subsequently unloaded.	Number	

3.2.4 JVM Garbage Collections Test

Manual memory management is time consuming, and error prone. Most programs still contain leaks. This is all doubly true with programs using exception-handling and/or threads. Garbage collection (GC) is a part of a Java application's JVM that automatically determines what memory a program is no longer using, and recycles it for other use. It is also known as "automatic storage (or memory) reclamation". The JVM Garbage Collections test reports the performance statistics pertaining to the JVM's garbage collection.

Target of the test : A Java application

Agent deploying the test : An internal/remote agent

Outputs of the test : One set of results for each garbage collector that is reclaiming the unused memory on the JVM of the Java application being monitored

Configurable parameters for the test

Parameter	Description
Test period	How often should the test be executed
Host	The host for which the test is to be configured.
Port	The port number at which the specified host listens to
Mode	This test can extract metrics from the Java application using either of the following mechanisms: • Using SNMP-based access to the Java runtime MIB statistics; • By contacting the Java runtime (JRE) of the application via JMX To configure the test to use SNMP, select the SNMP option. On the other hand, choose the JMX option to configure the test to use JMX instead. By default, the JMX option is chosen here.
JMX remote port	This parameter appers only if the mode is set to jmx. Here, specify the port at which the jmx listens for requests from remote hosts. Ensure that you specify the same port that you configured in the management.properties file in the <JAVA_ HOME>\jre\lib\management folder used by the target application (see page **3**).
JNDIname	This parameter appears only if the mode is set to JMX. The JNDIname is a lookup name for connecting to the JMX connector. By default, this is jmxrmi. If you have resgistered the JMX connector in the RMI registery using a different lookup name, then

Parameter	Description
	you can change this default value to reflect the same.
User, Password, and Confirm password	These parameters appear only if the Mode is set to JMX. If JMX requires authentication only (but no security), then ensure that the user and password parameters are configured with the credentials of a user with read-write access to JMX. To know how to create this user, refer to Section **2.1.2**. Confirm the password by retyping it in the confirm password text box.
Provider	This parameter appears only if the Mode is set to JMX. This test uses a JMX Provider to access the MBean attributes of the target Java application and collect metrics. Specify the package name of this JMX Provider here. By default, this is set to com.sun.jmx.remote.protocol.
Timeout	Specify the duration (in seconds) for which this test should wait for a response from the target Java application. If there is no response from the target beyond the configured duration, the test will timeout. By default, this is set to 240 seconds if the Mode is JMX, and 10 seconds if the Mode is SNMP.
SNMPPort	This parameter appears only if the Mode is set to SNMP. Here specify the port number through which the server exposes its SNMP MIB. Ensure that you specify the same port you configured in the management.properties file in the <JAVA_ HOME>\jre\lib\management folder used by the target application (see page **18**).
SNMP Version	This parameter appears only if the Mode is set to SNMP. The default selection in the SNMP version list is v1. However, for this test to work, you have to select SNMP v2 or v3 from this list, depending upon which version of SNMP is in use in the target environment.
SNMP Community	This parameter appears only if the Mode is set to SNMP. Here, specify the SNMP community name that the test uses to communicate with the mail server. The default is public. This parameter is specific to SNMP v1 and v2 only. Therefore, if the SNMP version chosen is v3, then this parameter will not appear.
User name	This parameter appears only when v3 is selected as the SNMP version. SNMP version 3 (SNMPv3) is an extensible SNMP Framework which supplements the SNMPv2 Framework, by additionally supporting message security, access control, and remote SNMP configuration capabilities. To extract performance statistics from the MIB using the highly secure SNMP v3 protocol, the eG agent has to be configured with the required access privileges – in other words, the eG agent should connect to the MIB using the credentials of a user with access permissions to be MIB. Therefore, specify the name of such a user against this parameter.
Context	This parameter appears only when v3 is selected as the SNMPVERSION. An SNMP context is a collection of management information accessible by an SNMP entity. An

Parameter	Description
	item of management information may exist in more than one context and an SNMP entity potentially has access to many contexts. A context is identified by the SNMPEngineID value of the entity hosting the management information (also called a contextEngineID) and a context name that identifies the specific context (also called a contextName). If the USERNAME provided is associated with a context name, then the eG agent will be able to poll the MIB and collect metrics only if it is configured with the context name as well. In such cases therefore, specify the context name of the username in the context text box. By default, this parameter is set to none.
Authpass	Specify the password that corresponds to the above-mentioned user name. This parameter once again appears only if the snmpversion selected is v3.
Confirm password	Confirm the Authpass by retyping it here
Authtype	This parameter too appears only if v3 is selected as the snmpversion. From the authtype list box, choose the authentication algorithm using which SNMP v3 converts the specified username and password into a 32-bit format to ensure security of SNMP transactions. You can choose between the following options: • MD5 – Message Digest Algorithm • SHA – Secure Hash Algorithm
Encryptflag	This flag appears only when v3 is selected as the snmpversion. By default, the eG agent does not encrypt SNMP requests. Accordingly, the flag is set to No by default. To ensure that SNMP requests sent by the eG agent are encrypted, select the Yes option.
Encrypttype	If the Encryptflag is set to Yes, then you will have to mention the encryption type by selecting an option from the Encrypttype list. SNMP v3 supports the following encryption types: • DES – Data Encryption Standard • AES – Advanced Encryption Standard
Encryptpassword	Specify the encryption password here.
Confirm password	Confirm the encryption password by retyping it here.
Data over TCP	This parameter is applicable only if mode is set to SNMP. By default, in an IT environment, all data transmission occurs over UDP. Some environments however, may be specifically configured to offload a fraction of the data traffic – for instance, certain types of data traffic or traffic pertaining to specific components – to other protocols like TCP, so as to prevent UDP overloads. In such environments, you can

Parameter	Description
	instruct the eG agent to conduct the SNMP data traffic related to the monitored target over TCP (and not UDP). For this, set this flag to Yes. By default, this flag is set to No.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
No of garbage collections started	Indicates the number of times this garbage collector was started to release dead objects from memory during the last measurement period.	Number	
Time taken for garbage collection	Indicates the time taken to by this garbage collector to perform the current garbage collection operation.	Secs	Ideally, the value of both these measures should be low. This is because, the garbage collection (GC) activity tends to suspend the operations of the application until such time that GC ends. Longer the GC time, longer it would take for the application to resume its functions. To minimize the impact of GC on application performance, it is best to ensure that GC activity does not take too long to complete.
Percent of time spent by JVM for garbage collection	Indicates the percentage of time spent by this garbage collector on garbage collection during the last measurement period.	Percent	

3.2.5 JVM Memory Pool Garbage Collections Test

While the **JVM Garbage Collections** test reports statistics indicating how well each collector on the JVM performs garbage collection, the measures reported by the **JVM Memory Pool Garbage Collections** test help assess the impact of the garbage collection activity on the availability and usage of memory in each memory pool of the JVM. Besides revealing the count of garbage collections per collector and the time taken by each collector to perform garbage collection on the individual memory pools, the test also compares the amount of memory used and available for use pre and post garbage collection in each of the memory pools. This way, the test enables

administrators to guage the effectiveness of the garbage collection activity on the memory pools, and helps them accurately identify those memory pools where enough memory could not reclaimed or where the garbage collectors spent too much time.

Note:

- This test will work only if the target Java application uses the JDK/JRE offered by one of the following vendors: Oracle, Sun, OpenJDK. IBM JDK/JRE is not supported.
- This test will not report metrics if the Mode parameter of the test is set to **SNMP**.

Target of the test : A Java application

Agent deploying the test : An internal/remote agent

Outputs of the test : One set of results for every GarbageCollector:MemoryPool pair on the JVM of the Java application being monitored

Configurable parameters for the test

Parameter	Description
Test period	How often should the test be executed.
Host	The host for which the test is to be configured.
Port	The port number at which the specified host listens to.
Measure Mode	This test allows you the option to collect the desired metrics using one of the following methodologies: • By contacting the Java runtime (JRE) of the application via JMX • Using GC logs To use JMX for metrics collections, set the measure modeto JMX. On the other hand, if you intend to use the GC log files for collecting the required metrics, set the measure mode to Log File. In this case, you would be required to enable GC logging. The procedure for this has been detailed in Section **3.2.5.1**.
JMX remote port	This parameter appers only if the mode is set to jmx. Here, specify the port at which the jmx listens for requests from remote hosts. Ensure that you specify the same port that you configured in the management.properties file in the <JAVA_ HOME>\jre\lib\management folder used by the target application (see page **3**).
JNDIname	This parameter appears only if the mode is set to JMX. The JNDIname is a lookup name for connecting to the JMX connector. By default, this is jmxrmi. If you have

Parameter	Description
	resgistered the JMX connector in the RMI registery using a different lookup name, then you can change this default value to reflect the same.
User, Password, and Confirm password	These parameters appear only if the Mode is set to JMX. If JMX requires authentication only (but no security), then ensure that the user and password parameters are configured with the credentials of a user with read-write access to JMX. To know how to create this user, refer to Section **2.1.2**. Confirm the password by retyping it in the confirm password text box.
JREHome	This parameter will be available only if the Measure Mode is set to Log File. Specify the full path to the Java Runtime Environment (JRE) used by the target application.
Logfilename	This parameter will be available only if the Measure Mode is set to Log File. Specify the full path to the GC log file to be used for metrics collection.
Provider	This parameter appears only if the Mode is set to JMX. This test uses a JMX Provider to access the MBean attributes of the target Java application and collect metrics. Specify the package name of this JMX Provider here. By default, this is set to com.sun.jmx.remote.protocol.
Timeout	Specify the duration (in seconds) for which this test should wait for a response from the target Java application. If there is no response from the target beyond the configured duration, the test will timeout. By default, this is set to 240 seconds if the Mode is JMX, and 10 seconds if the Mode is SNMP.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Has garbage collection happened?	Indicates whether garbage collection occurred on this memory pool in the last measurement period.		This measure reports the value Yes if garbage collection took place or No if it did not take place on the memory pool. The numeric values that correspond to the measure values of Yes and No are listed below: State: Yes — Value: 1 State: No — Value: 0 **Note:**

Measurement	Description	Measurement Unit	Interpretation
			By default, this measure reports the value Yes or No to indicate whether a GC occurred on a memory pool or not. The graph of this measure however, represents the same using the numeric equivalents – 0 or 1.
Collection count	Indicates the number of time in the last measurement pool garbage collection was started on this memory pool.	Number	
Initial memory before GC	Indicates the initial amount of memory (in MB) that this memory pool requests from the operating system for memory management during startup, before GC process.	MB	Comparing the value of these two measures for a memory pool will give you a fair idea of the effectiveness of the garbage collection activity. If garbage collection reclaims a large amount of memory from the memory pool, then the Initial memory after GC will drop. On the other hand, if the garbage collector does not reclaim much memory from a memory pool, or if the Java application suddenly runs a memory-intensive process when GC is being performed, then the Initial memory after GC may be higher than the Initial memory before GC.
Initial memory after GC	Indicates the initial amount of memory (in MB) that this memory pool requests from the operating system for memory management during startup, after GC process	MB	
Max memory before GC	Indicates the maximum amount of memory that can be used for memory management by this memory pool, before GC process.	MB	Comparing the value of these two measures for a memory pool will provide you with insights into the effectiveness of the garbage collection activity. If garbage collection reclaims a large amount of memory from the memory pool, then the Max memory after GC will drop. On the other hand, if the garbage collector does not reclaim much memory from a memory pool, or

Measurement	Description	Measurement Unit	Interpretation
Max memory after GC	Indicates the maximum amount of memory (in MB) that can be used for memory management by this pool, after the GC process.	MB	if the Java application suddenly runs a memory-intensive process when GC is being performed, then the Max memory after GC value may exceed the Max memory before GC.
Committed memory before GC	Indicates the amount of memory that is guaranteed to be available for use by this memory pool, before the GC process.	MB	
Committed memory after GC	Indicates the amount of memory that is guaranteed to be available for use by this memory pool, after the GC process.	MB	
Used memory before GC	Indicates the amount of memory used by this memory pool before GC.	MB	Comparing the value of these two measures for a memory pool will provide you with insights into the effectiveness of the garbage collection activity. If garbage collection reclaims a large amount of memory from the memory pool, then the Used memory after GC may drop lower than the Used memory before GC. On the other hand, if the garbage collector does not reclaim much memory from a memory pool, or if the Java application suddenly runs a memory-intensive process when GC is being performed, then the Used memory after GC value may exceed the Used memory before GC.
Used memory after GC	Indicates the amount of memory used by this memory pool after GC.	MB	
Percentage of memory collected	Indicates the percentage of memory collected from this pool by the GC activity.	Percent	A high value for this measure is indicative of a large amount of unused memory in the pool. A low value on the

Measurement	Description	Measurement Unit	Interpretation
			other hand indicates that the memory pool has been over-utilized. Compare the value of this measure across pools to identify the pools that have very little free memory. If too many pools appear to be running short of memory, it could indicate that the target application is consuming too much memory, which in the long run, can slow down the application significantly.
Collection duration	Indicates the time taken by this garbage collector for collecting unused memory from this pool.	Mins	Ideally, the value of this measure should be low. This is because, the garbage collection (GC) activity tends to suspend the operations of the application until such time that GC ends. Longer the GC time, longer it would take for the application to resume its functions. To minimize the impact of GC on application performance, it is best to ensure that GC activity does not take too long to complete.

3.2.5.1 Enabling GC Logging

If you want the **JVM Memory Pools Garbage Collections** test to use the GC log file to report metrics, then, you first need to enable GC logging. For this, follow the steps below:

1. Edit the startup script file of the Java application being monitored. Figure 20 depicts the startup script file of a sample application.

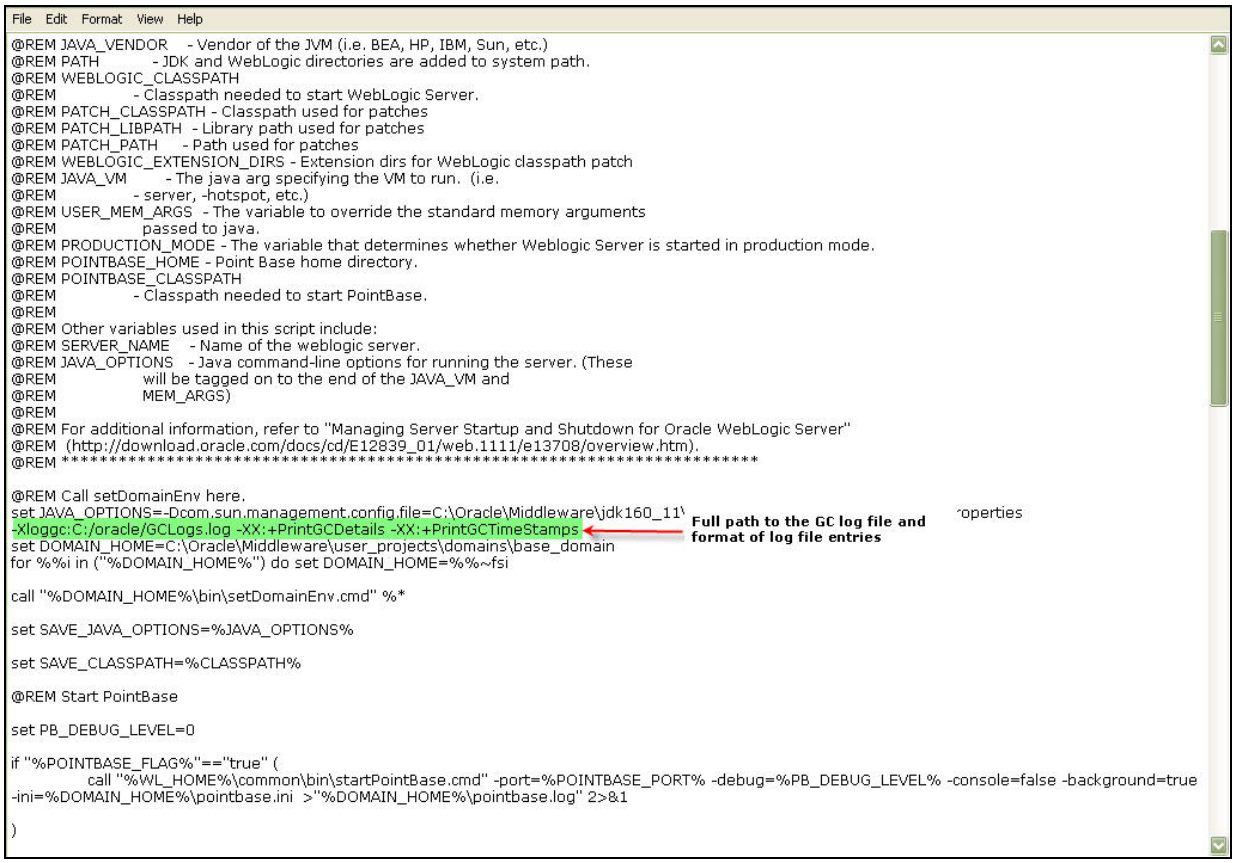


Figure 3.3: Editing the startup script file of a sample Java application

2. Add the line indicated by Figure 20 to the startup script file. This line should be of the following format:

```
-Xloggc:<Full path to the GC log file to which GC details are to be logged> -
XX:+PrintGCDetails -XX:+PrintGCTimeStamps
```

Here, the entry, -XX:+PrintGCDetails -XX:+PrintGCTimeStamps, refers to the format in which GC details are to be logged in the specified log file. Note that this test can monitor only those GC log files which contain log entries of this format.

3. Finally, save the file and restart the application.

3.2.6 JVM Threads Test

This test reports the status of threads running in the JVM. Details of this test can be used to identify resource-hungry threads.

Note:

If the Mode parameter of this test is set to **SNMP**, then stack trace will not be available. Also, detailed diagnostics will not report *CPU Time*.

Target of the test : A Java application

Agent deploying the test : An internal/remote agent

Outputs of the test : One set of results for the Java application being monitored

Configurable parameters for the test

Parameter	Description
Test period	How often should the test be executed
Host	The host for which the test is to be configured.
Port	The port number at which the specified host listens to
Mode	This test can extract metrics from the Java application using either of the following mechanisms: • Using SNMP-based access to the Java runtime MIB statistics; • By contacting the Java runtime (JRE) of the application via JMX To configure the test to use SNMP, select the SNMP option. On the other hand, choose the JMX option to configure the test to use JMX instead. By default, the JMX option is chosen here.
JMX Remote Port	This parameter appers only if the mode is set to jmx. Here, specify the port at which the jmx listens for requests from remote hosts. Ensure that you specify the same port that you configured in the management.properties file in the <JAVA_HOME>\jre\lib\management folder used by the target application (see page **3**).
JNDIname	This parameter appears only if the mode is set to JMX. The JNDIname is a lookup name for connecting to the JMX connector. By default, this is jmxrmi. If you have resgistered the JMX connector in the RMI registery using a different lookup name, then you can change this default value to reflect the same.
User, Password, and Confirm password	These parameters appear only if the Mode is set to JMX. If JMX requires authentication only (but no security), then ensure that the user and password parameters are configured with the credentials of a user with read-write access to JMX. To know how to create this user, refer to Section **2.1.2**. Confirm the

Parameter	Description
	password by retyping it in the confirm password text box.
Provider	This parameter appears only if the Mode is set to JMX. This test uses a JMX Provider to access the MBean attributes of the target Java application and collect metrics. Specify the package name of this JMX Provider here. By default, this is set to com.sun.jmx.remote.protocol.
Timeout	Specify the duration (in seconds) for which this test should wait for a response from the target Java application. If there is no response from the target beyond the configured duration, the test will timeout. By default, this is set to 240 seconds if the Mode is JMX, and 10 seconds if the Mode is SNMP.
SNMPPort	This parameter appears only if the Mode is set to SNMP. Here specify the port number through which the server exposes its SNMP MIB. Ensure that you specify the same port you configured in the management.properties file in the <JAVA_HOME>\jre\lib\management folder used by the target application (see page **18**).
SNMP Version	This parameter appears only if the Mode is set to SNMP. The default selection in the SNMP version list is v1. However, for this test to work, you have to select SNMP v2 or v3 from this list, depending upon which version of SNMP is in use in the target environment.
SNMP Community	This parameter appears only if the Mode is set to SNMP. Here, specify the SNMP community name that the test uses to communicate with the mail server. The default is public. This parameter is specific to SNMP v1 and v2 only. Therefore, if the SNMP version chosen is v3, then this parameter will not appear.
User name	This parameter appears only when v3 is selected as the SNMP version. SNMP version 3 (SNMPv3) is an extensible SNMP Framework which supplements the SNMPv2 Framework, by additionally supporting message security, access control, and remote SNMP configuration capabilities. To extract performance statistics from the MIB using the highly secure SNMP v3 protocol, the eG agent has to be configured with the required access privileges – in other words, the eG agent should connect to the MIB using the credentials of a user with access permissions to be MIB. Therefore, specify the name of such a user against this parameter.
Context	This parameter appears only when v3 is selected as the SNMPVERSION. An

Parameter	Description
	SNMP context is a collection of management information accessible by an SNMP entity. An item of management information may exist in more than one context and an SNMP entity potentially has access to many contexts. A context is identified by the SNMPEngineID value of the entity hosting the management information (also called a contextEngineID) and a context name that identifies the specific context (also called a contextName). If the USERNAME provided is associated with a context name, then the eG agent will be able to poll the MIB and collect metrics only if it is configured with the context name as well. In such cases therefore, specify the context name of the username in the context text box. By default, this parameter is set to none.
Authpass	Specify the password that corresponds to the above-mentioned user name. This parameter once again appears only if the snmpversion selected is v3.
Confirm password	Confirm the Authpass by retyping it here
Authtype	This parameter too appears only if v3 is selected as the SNMPversion. From the Authtype list box, choose the authentication algorithm using which SNMP v3 converts the specified username and password into a 32-bit format to ensure security of SNMP transactions. You can choose between the following options: • MD5 – Message Digest Algorithm • SHA – Secure Hash Algorithm
Encryptflag	This flag appears only when v3 is selected as the SNMPversion. By default, the eG agent does not encrypt SNMP requests. Accordingly, the flag is set to No by default. To ensure that SNMP requests sent by the eG agent are encrypted, select the Yes option.
Encrypttype	If the Encryptflag is set to Yes, then you will have to mention the encryption type by selecting an option from the Encrypttype list. SNMP v3 supports the following encryption types: • DES – Data Encryption Standard • AES – Advanced Encryption Standard
Encryptpassword	Specify the encryption password here.
Confirm password	Confirm the encryption password by retyping it here.

Parameter	Description
PCT Medium CPU Util Threads	By default, this parameter is set to 50. This implies that, by default, the threads for which the current CPU consumption is between 50% and 70% (the default value of the pct high cpu util threads parameter) will be counted as medium CPU-consuming threads. The count of such threads will be reported as the value of the *Medium CPU threads* measure. This default setting also denotes that threads that consume less than 50% CPU will, by default, be counted as Low CPU threads. If need be, you can modify the value of this parameter to change how much CPU should be used by a thread for it to qualify as a medium CPU-consuming thread. This will consequently alter the count of low CPU-consuming threads as well.
PCT High CPU Util Threads	By default, this parameter is set to 70. This implies that, by default, the threads that are currently consuming over 70% of CPU time are counted as high CPU consumers. The count of such threads will be reported as the value of the *High CPU threads* measure. If need be, you can modify the value of this parameter to change how much CPU should be used by a thread for it to qualify as a high CPU-consuming thread.
Max Thread Count	By default, this parameter is set to 20. This implies that the detailed diagnosis of the Runnable threads, Waiting threads, and *Timed waiting threads* measures will by default display only the top-20 JVM threads in terms of CPU consumption. To view more threads as part of detailed diagnostics, increase the value of this parameter. To view all threads that are in the said state (eg., runnable, waiting, and timed waiting), specify All or * against this parameter.
USEPS	This flag is applicable only for AIX LPARs. By default, on AIX LPARs, this test uses the tprof command to compute CPU usage. Accordingly, this flag is set to No by default. On some AIX LPARs however, the tprof command may not function properly (this is an AIX issue). While monitoring such AIX LPARs therefore, you can configure the test to use the ps command instead for metrics collection. To do so, set this flag to Yes. **Note:** Alternatively, you can set the AIXUSEPS flag in the [AGENT_SETTINGS] section of the *eg_tests.ini* file (in the <EG_INSTALL_SIR>\manager\config directory) to yes (default: no) to enable the eG agent to use the ps command for CPU usage computations on AIX LPARs. If this global flag and the USEPS flag for a specific component are both set to no, then the test will use the default tprof command to compute CPU usage for AIX LPARs. If either of these flags is

Parameter	Description
	set to yes, then the ps command will perform the CPU usage computations for monitored AIX LPARs. In some high-security environments, the tprof command may require some special privileges to execute on an AIX LPAR (eg., sudo may need to be used to run tprof). In such cases, you can prefix the tprof command with another command (like sudo) or the full path to a script that grants the required privileges to tprof. To achieve this, edit the eg_tests.ini file (in the <EG_INSTALL_DIR>\manager\config directory), and provide the prefix of your choice against the *AixTprofPrefix* parameter in the [AGENT_SETTINGS] section. Finally, save the file. For instance, if you set the *AixTprofPrefix* parameter to sudo, then the eG agent will call the tprof command as sudo tprof.
Data over TCP	This parameter is applicable only if mode is set to SNMP. By default, in an IT environment, all data transmission occurs over UDP. Some environments however, may be specifically configured to offload a fraction of the data traffic – for instance, certain types of data traffic or traffic pertaining to specific components – to other protocols like TCP, so as to prevent UDP overloads. In such environments, you can instruct the eG agent to conduct the SNMP data traffic related to the monitored target over TCP (and not UDP). For this, set this flag to Yes. By default, this flag is set to No.
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 this parameter.
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

Parameter	Description
	• Both the normal and abnormal frequencies configured for the detailed diagnosis measures should not be 0.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Total threads	Indicates the total number of threads (including daemon and non-daemon threads).	Number	
Runnable threads	Indicates the current number of threads in a runnable state.	Number	The detailed diagnosis of this measure, if enabled, lists the names of the top- 20 (default) runnable threads in terms of their CPU usage. The time for which the thread was in a blocked state, waiting state, etc., are provided as part of the detailed diagnostics. You can change the sort order to view threads by waiting time, blocked time, etc.
Blocked threads	Indicates the number of threads that are currently in a blocked state.	Number	If a thread is trying to take a lock (to enter a synchronized block), but the lock is already held by another thread, then such a thread is called a blocked thread. The detailed diagnosis of this measure, if enabled, provides in-depth information related to all the blocked threads.
Waiting threads	Indicates the number of threads that are currently in a waiting state.	Number	A thread is said to be in a Waiting state if the thread enters a synchronized block, tries to take a lock that is already held by another

Measurement	Description	Measurement Unit	Interpretation
			thread, and hence, waits till the other thread notifies that it has released the lock. Ideally, the value of this measure should be low. A very high value could be indicative of excessive waiting activity on the JVM. You can use the detailed diagnosis of this measure, if enabled, to figure out which threads are currently in the waiting state. By default, the top-20 waiting threads in terms of CPU usage will be listed. You can change the sort order to view threads by waiting time, blocked time, etc. While waiting, the Java application program does no productive work and its ability to complete the task-at-hand is degraded. A certain amount of waiting may be acceptable for Java application programs. However, when the amount of time spent waiting becomes excessive or if the number of times that waits occur exceeds a reasonable amount, the Java application program may not be programmed correctly to take advantage of the available resources. When this happens, the delay caused by the waiting Java application programs elongates the response time experienced by an end user. An enterprise may use

Measurement	Description	Measurement Unit	Interpretation
			Java application programs to perform various functions. Delays based on abnormal degradation consume employee time and may be costly to corporations.
Timed waiting threads	Indicates the number of threads in a *TIMED_ WAITING* state.	Number	When a thread is in the TIMED_ WAITING state, it implies that the thread is waiting for another thread to do something, but will give up after a specified time out period. To view the details of threads in the *TIMED_ WAITING* state, use the detailed diagnosis of this measure, if enabled. By default, the top-20 timed waiting threads in terms of CPU usage will be listed. You can change the sort order to view threads by waiting time, blocked time, etc.
Low CPU threads	Indicates the number of threads that are currently consuming CPU lower than the value configured in the PCT Medium CPU Util Threads text box.	Number	To know which threads are consuming low CPU, use the detailed diagnosis of this measure.
Medium CPU threads	Indicates the number of threads that are currently consuming CPU that is higher than the value configured in the PCT Medimum CPU Util Threads text box and is lower than or	Number	To know which threads are consuming medium CPU, use the detailed diagnosis of this measure.

Measurement	Description	Measurement Unit	Interpretation
	equal to the value specified in the PCT High CPU Util Threads text box.		
High CPU threads	Indicates the number of threads that are currently consuming CPU that is greater than the percentage configured in the PCT High CPU Util Threads text box.	Number	Ideally, the value of this measure should be very low. A high value is indicative of a resource contention at the JVM. Under such circumstances, you might want to identify the resource- hungry threads. To know which threads are consuming excessive CPU, use the detailed diagnosis of this measure.
Peak threads	Indicates the highest number of live threads since JVM started.	Number	
Total threads	Indicates the the total number of threads started (including daemon, non- daemon, and terminated) since JVM started.	Number	
Daemon threads	Indicates the current number of live daemon threads.	Number	
Deadlock threads	Indicates the current number of deadlocked threads.	Number	Ideally, this value should be 0. A high value is a cause for concern, as it indicates that many threads are blocking one another causing the application performance to suffer. The detailed diagnosis of this measure, if enabled, lists the deadlocked threads and their resource usage.

Note:

If the mode for the **JVM Threads** test is set to SNMP, then the detailed diagnosis of this test will not display the **Blocked Time** and **Waited Time** for the threads. To make sure that detailed diagnosis reports these details also, do the following:

- Login to the application host.
- Go to the <JAVA_HOME>\jre\lib\management folder used by the target application, and edit the management.properties file in that folder.
- Append the following line to the file:

```
com.sun.management.enableThreadContentionMonitoring
```

- Finally, save the file.

3.2.6.1 Accessing Stack Trace using the STACK TRACE link in the Measurements Panel

While viewing the measures reported by the **JVM Thread** test, you can also view the resource usage details and the stack trace information for all the threads, by clicking on the stack trace link in the **Measurements** panel.

Note:

If the mode set for the **JVM Thread** test is SNMP, the stack trace details may not be available.

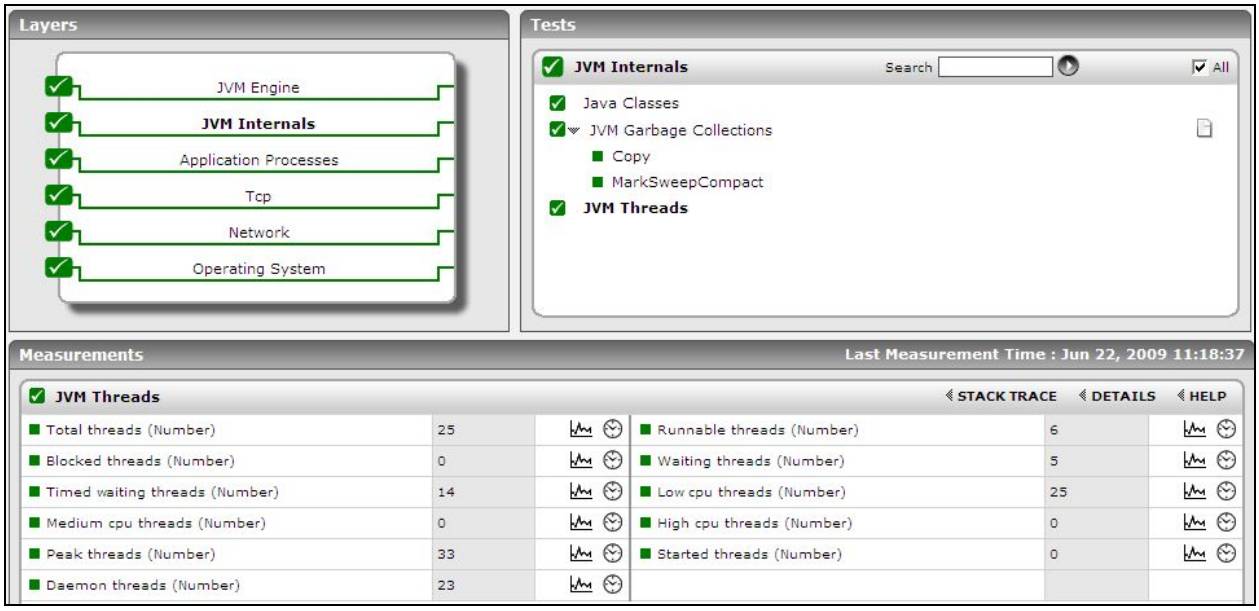


Figure 3.4: The STACK TRACE link

A stack trace (also called stack backtrace or stack traceback) is a report of the active stack frames instantiated by the execution of a program. It is commonly used to determine what threads are currently active in the JVM, and which threads are in each of the different states – i.e., alive, blocked, waiting, timed waiting, etc.

Typically, when a Java application begins exhibiting erratic resource usage patterns, it often takes administrators hours, even days to figure out what is causing this anomaly – could it be owing to one/more resource-intensive threads being executed by the application? If so, what is causing the thread to erode resources? Is it an inefficient piece of code? In which case, which line of code could be the most likely cause for the spike in resource usage? To be able to answer these questions accurately, administrators need to know the complete list of threads that the application executes, view the stack trace of each thread, analyze each stack trace in a top-down manner, and trace where the problem originated.

eG Enterprise simplifies this seemingly laborious procedure by not only alerting administrators instantly to excessive resource usage by a target application, but also by automatically identifying the problematic thread(s), and providing the administrator with quick and easy access to the stack trace information of that thread; with the help of stack trace, administrators can effortlessly drill down to the exact line of code that requires optimization.

To access the stack trace information of a thread, click on the **STACK TRACE** link in the **Measurements** panel of Figure 3.4.

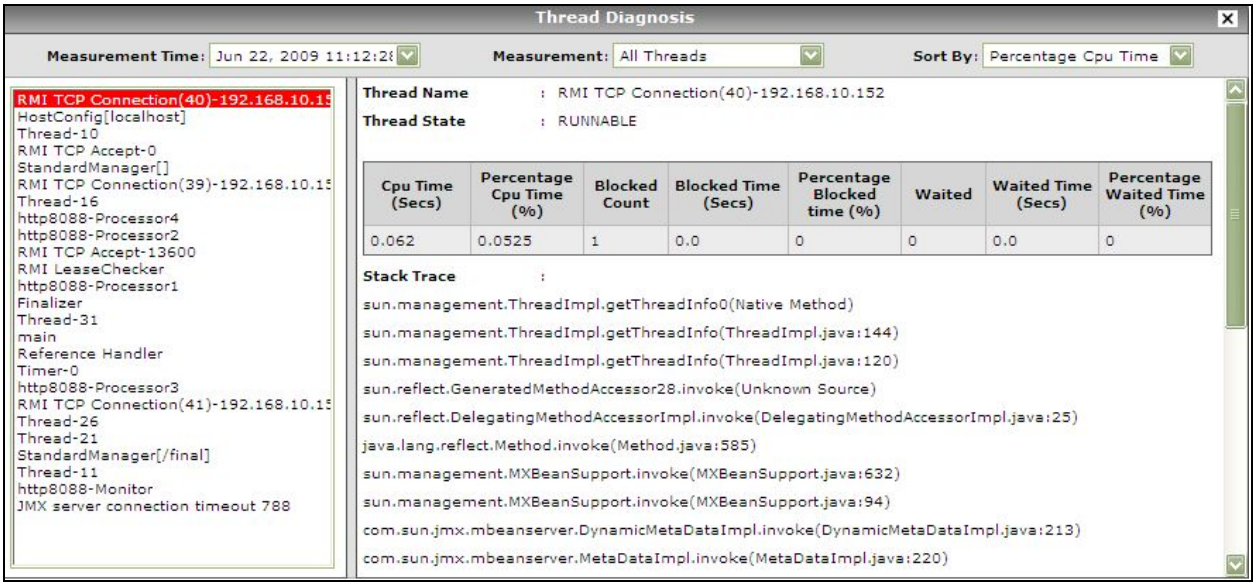


Figure 3.5: Stack trace of a resource-intensive thread

Figure 3.5 that appears comprises of two panels. The left panel, by default, lists all the threads that the target application executes, starting with the threads that are currently live. Accordingly, the **All Threads** option is chosen by default from the **Measurement** list. If need be, you can override the default setting by choosing a different option from the **Measurement** list – in other words, instead of viewing the complete list of threads, you can choose to view threads of a particular type or which are in a particular state alone in Figure 3.5, by selecting a different **Measurement** from Figure 3.5. For instance, to ensure that the left panel displays only those threads that are currently in a runnable state, select the **Live threads** option from the **Measurement** list. The contents of the left panel will change as depicted by Figure 3.6.

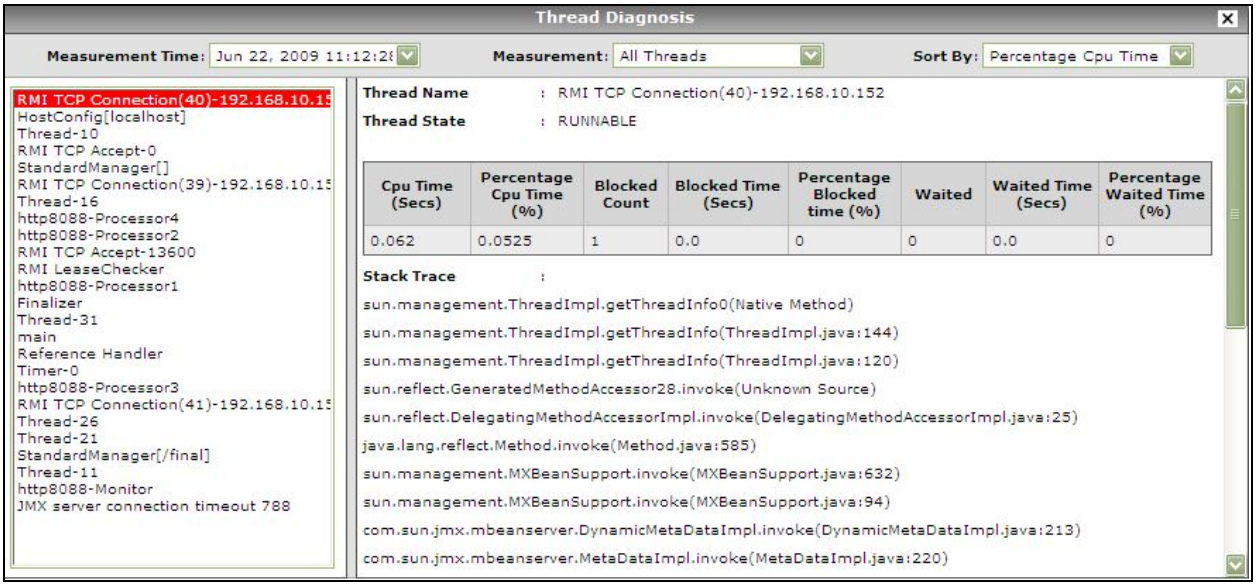


Figure 3.6: Thread diagnosis of live threads

Also, the thread list in the left panel is by default sorted in the descending order of the Percent CPU Time of the threads. This implies that, by default, the first thread in the list will be the thread that is currently active and consuming the maximum CPU. You can change the sort order by selecting a different option from the **Sort by** list in Figure 3.6.

Typically, the contents of the right panel change according to the thread chosen from the left. Since the first thread is the default selection in the left panel, and this thread by default consumes the maximum CPU, we can conclude that the right panel will by default display the details of the leading CPU consumer. Besides the name and state of the chosen thread, the right panel will provide the following information:

- **Cpu Time** : The amount of CPU processing time (in seconds) consumed by the thread during the last measurement period;

- **Percent Cpu Time**: The percentage of time the thread was using the CPU during the last measurement period;
- **Blocked Count**: The number of the times during the last measurement period the thread was blocked waiting for another thread;
- **Blocked Time**: The total duration for which the thread was blocked during the last measurement period;
- **Percentage Blocked Time**: The percentage of time (in seconds) for which the thread was blocked during the last measurement period;
- **Waited**: The number of times during the last measurement period the thread was waiting for some event to happen (eg., wait for a thread to finish, wait for a timing event to finish, etc.);
- **Waited Time**: The total duration (in seconds) for which the thread was waiting during the last measurement period;
- **Percentage Waited Time**: The percentage of time for which the thread was waiting during the last measurement period.

In addition to the above details, the right panel provides the **Stack Trace** of the thread.

In the event of a sudden surge in the CPU usage of the target Java application, the **Thread Diagnosis** window of Figure 3.6 will lead you to the CPU-intensive thread, and will also provide you with the **Stack Trace** of that thread. By analyzing the stack trace in a top-down manner, you can figure out which method/routine called which, and thus locate the exact line of code that could have contributed to the sudden CPU spike.

If the CPU usage has been increasing over a period of time, then, you might have to analyze the stack trace for one/more prior periods, so as to perform accurate root-cause diagnosis. By default, the **Thread Diagnosis** window of Figure 3.6 provides the stack trace for the current measurement period only. If you want to view the stack trace for a previous measurement period, you will just have to select a different option from the **Measurement Time** list. By reviewing the code executed by a thread for different measurement periods, you can figure out out if the same line of code is responsible for the increase in CPU usage.

3.3 The JVM Engine Layer

The JVM Engine layer measures the overall health of the JVM engine by reporting statistics related to the following:

- The CPU usage by the engine
- How the JVM engine manages memory
- The uptime of the engine

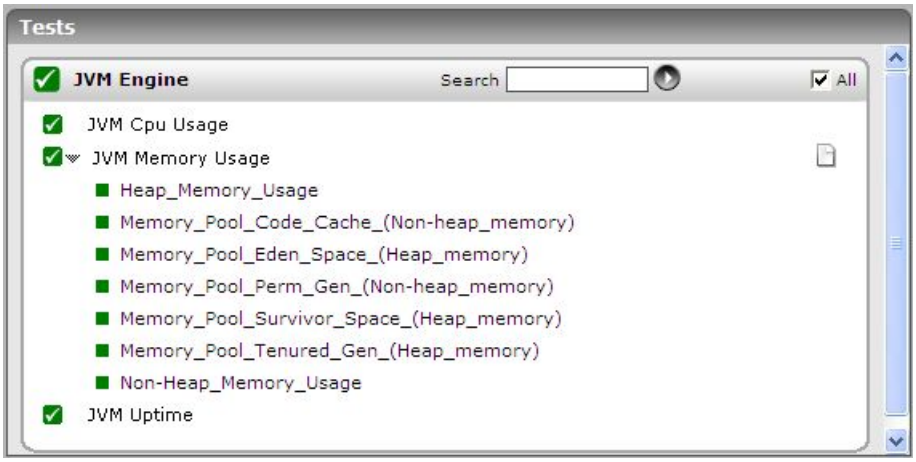


Figure 3.7: The tests associated with the JVM Engine layer

3.3.1 JVM CPU Usage Test

This test measures the CPU utilization of the JVM. If the JVM experiences abnormal CPU usage levels, you can use this test to instantly drill down to the threads that are contributing to the CPU spike. Detailed stack trace information provides insights to code level information that can highlight problems with the design of the Java application.

Note:

This test will not report metrics if the Mode parameter of the test is set to **SNMP**.

Target of the test : A Java application

Agent deploying the test : An internal/remote agent

Outputs of the test : One set of results for the Java application being monitored

Configurable parameters for the test

Parameter	Description
Test period	How often should the test be executed
Host	The host for which the test is to be configured.

Parameter	Description
Port	The port number at which the specified host listens to
Mode	This test can extract metrics from the Java application using either of the following mechanisms: • Using SNMP-based access to the Java runtime MIB statistics; • By contacting the Java runtime (JRE) of the application via JMX To configure the test to use SNMP, select the SNMP option. On the other hand, choose the JMX option to configure the test to use JMX instead. By default, the JMX option is chosen here.
JMX remote port	This parameter appers only if the mode is set to jmx. Here, specify the port at which the jmx listens for requests from remote hosts. Ensure that you specify the same port that you configured in the management.properties file in the <JAVA_HOME>\jre\lib\management folder used by the target application (see page **3**).
JNDIname	This parameter appears only if the mode is set to JMX. The JNDIname is a lookup name for connecting to the JMX connector. By default, this is jmxrmi. If you have resgistered the JMX connector in the RMI registery using a different lookup name, then you can change this default value to reflect the same.
User, Password, and Confirm password	These parameters appear only if the Mode is set to JMX. If JMX requires authentication only (but no security), then ensure that the user and password parameters are configured with the credentials of a user with read-write access to JMX. To know how to create this user, refer to Section **2.1.2**. Confirm the password by retyping it in the Confirm Password text box.
Provider	This parameter appears only if the Mode is set to JMX. This test uses a JMX Provider to access the MBean attributes of the target Java application and collect metrics. Specify the package name of this JMX Provider here. By default, this is set to com.sun.jmx.remote.protocol.
Timeout	Specify the duration (in seconds) for which this test should wait for a response from the target Java application. If there is no response from the target beyond the configured duration, the test will timeout. By default, this is set to 240 seconds if the Mode is JMX, and 10 seconds if the Mode is SNMP.
SNMPPort	This parameter appears only if the Mode is set to SNMP. Here specify the port

Parameter	Description
	number through which the server exposes its SNMP MIB. Ensure that you specify the same port you configured in the management.properties file in the <JAVA_HOME>\jre\lib\management folder used by the target application (see page **18**).
SNMP Version	This parameter appears only if the Mode is set to SNMP. The default selection in the SNMP version list is v1. However, for this test to work, you have to select SNMP v2 or v3 from this list, depending upon which version of SNMP is in use in the target environment.
SNMP Community	This parameter appears only if the Mode is set to SNMP. Here, specify the SNMP community name that the test uses to communicate with the mail server. The default is public. This parameter is specific to SNMP v1 and v2 only. Therefore, if the SNMP version chosen is v3, then this parameter will not appear.
User name	This parameter appears only when v3 is selected as the SNMP version. SNMP version 3 (SNMPv3) is an extensible SNMP Framework which supplements the SNMPv2 Framework, by additionally supporting message security, access control, and remote SNMP configuration capabilities. To extract performance statistics from the MIB using the highly secure SNMP v3 protocol, the eG agent has to be configured with the required access privileges – in other words, the eG agent should connect to the MIB using the credentials of a user with access permissions to be MIB. Therefore, specify the name of such a user against this parameter.
Context	This parameter appears only when v3 is selected as the SNMPVERSION. An SNMP context is a collection of management information accessible by an SNMP entity. An item of management information may exist in more than one context and an SNMP entity potentially has access to many contexts. A context is identified by the SNMPEngineID value of the entity hosting the management information (also called a contextEngineID) and a context name that identifies the specific context (also called a contextName). If the USERNAME provided is associated with a context name, then the eG agent will be able to poll the MIB and collect metrics only if it is configured with the context name as well. In such cases therefore, specify the context name of the username in the context text box. By default, this parameter is set to none.
Authpass	Specify the password that corresponds to the above-mentioned user name.

Parameter	Description
	This parameter once again appears only if the snmpversion selected is v3.
Confirm password	Confirm the Authpass by retyping it here
Authtype	This parameter too appears only if v3 is selected as the snmpversion. From the authtype list box, choose the authentication algorithm using which SNMP v3 converts the specified username and password into a 32-bit format to ensure security of SNMP transactions. You can choose between the following options: • MD5 – Message Digest Algorithm • SHA – Secure Hash Algorithm
Encryptflag	This flag appears only when v3 is selected as the snmpversion. By default, the eG agent does not encrypt SNMP requests. Accordingly, the flag is set to No by default. To ensure that SNMP requests sent by the eG agent are encrypted, select the Yes option. • DES – Data Encryption Standard • AES – Advanced Encryption Standard
Encryptpassword	Specify the encryption password here.
Confirm password	Confirm the encryption password by retyping it here.
Data over TCP	This parameter is applicable only if mode is set to SNMP. By default, in an IT environment, all data transmission occurs over UDP. Some environments however, may be specifically configured to offload a fraction of the data traffic – for instance, certain types of data traffic or traffic pertaining to specific components – to other protocols like TCP, so as to prevent UDP overloads. In such environments, you can instruct the eG agent to conduct the SNMP data traffic related to the monitored target over TCP (and not UDP). For this, set this flag to Yes. By default, this flag is set to No.
USEPS	This flag is applicable only for AIX LPARs. By default, on AIX LPARs, this test uses the tprof command to compute CPU usage. Accordingly, this flag is set to No by default. On some AIX LPARs however, the tprof command may not function properly (this is an AIX issue). While monitoring such AIX LPARs therefore, you can configure the test to use the ps command instead for metrics collection. To do so, set this flag to Yes. **Note:**

Parameter	Description
	Alternatively, you can set the AIXUSEPS flag in the [AGENT_SETTINGS] section of the *eg_tests.ini* file (in the <EG_INSTALL_SIR>\manager\config directory) to yes (default: no) to enable the eG agent to use the ps command for CPU usage computations on AIX LPARs. If this global flag and the USEPS flag for a specific component are both set to no, then the test will use the default tprof command to compute CPU usage for AIX LPARs. If either of these flags is set to yes, then the ps command will perform the CPU usage computations for monitored AIX LPARs. In some high-security environments, the tprof command may require some special privileges to execute on an AIX LPAR (eg., sudo may need to be used to run tprof). In such cases, you can prefix the tprof command with another command (like sudo) or the full path to a script that grants the required privileges to tprof. To achieve this, edit the eg_tests.ini file (in the <EG_INSTALL_DIR>\manager\config directory), and provide the prefix of your choice against the *AixTprofPrefix* parameter in the [AGENT_SETTINGS] section. Finally, save the file. For instance, if you set the *AixTprofPrefix* parameter to sudo, then the eG agent will call the tprof command as sudo tprof.
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 this parameter.
Detailed Diagnosis	To make diagnosis more efficient and accurate, the eG Enterprise suite embeds an optional detailed diagnostic capability. With this capability, the eG agents can be configured to run detailed, more elaborate tests as and when specific problems are detected. To enable the detailed diagnosis capability of this test for a particular server, choose the **On** option. To disable the capability, click on the **Off** option. The option to selectively enable/disable the detailed diagnosis capability will be available only if the following conditions are fulfilled: • The eG manager license should allow the detailed diagnosis capability • Both the normal and abnormal frequencies configured for the detailed diagnosis measures should not be 0.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
CPU utilization of JVM	Indicates the percentage of total available CPU time taken up by the JVM.	Percent	If a system has multiple processors, this value is the total CPU time used by the JVM divided by the number of processors on the system. Ideally, this value should be low. An unusually high value or a consistent increase in this value is indicative of abnormal CPU usage, and could warrant further investigation. In such a situation, you can use the detailed diagnosis of this measure, if enabled, to determine which runnable threads are currently utilizing excessive CPU.

The detailed diagnosis of the CPU utilization of JVM measure lists all the CPU-consuming threads currently executing in the JVM, in the descending order of the Percentage CPU Time of the threads; this way, you can quickly and accurately identify CPU-intensive threads in the JVM. In addition to CPU usage information, the detailed diagnosis also reveals the following information for every thread:

- The number of times the thread was blocked during the last measurement period, the total duration of the blocks, and the percentage of time for which the thread was blocked;
- The number of times the thread was in waiting during the last measurement period, the total duration waited, and the percentage of time for which the thread waited;
- The **Stacktrace** of the thread, using which you can nail the exact line of code causing the CPU consumption of the thread to soar;

Details of the threads

Time	Thread Name	ThreadID	Thread State	Cpu Time (Secs)	Percentage Cpu Time (%)	Blocked Count	Blocked Time (Secs)	Percentage Blocked Time (%)	Waited	Waited Time (Secs)	Percentage Waited Time (%)	Stacktrace
Jun 22, 2009 14:42:30												Stack Trace
	http7077-Processor2	19	RUNNABLE	2.296	0.6651	103	0.006	0	83	534.26	18.18	java.net.SocketInputStream.socketRead0(Native Method); java.net.SocketInputStream.read(SocketInputStream.java:129); org.apache.coyote.http11.InternalInputBuffer.fill(InternalInputBuffer.java:767); org.apache.coyote.http11.InternalInputBuffer.parseRequestLine (InternalInputBuffer.java:428); org.apache.coyote.http11.Http11Processor.process (Http11Processor.java:790); org.apache.coyote.http11.Http11Protocol$Http11ConnectionHandler.processConnection (Http11Protocol.java:700); org.apache.tomcat.util.net.TcpWorkerThread.runIt (PoolTcpEndpoint.java:584); org.apache.tomcat.util.threads.ThreadPool$ControlRunnable.run (ThreadPool.java:683); java.lang.Thread.run(Thread.java:619);
	http7077-Processor4	21	RUNNABLE	2.89	0.4811	231	0.045	0	403	654.029	0	java.net.SocketInputStream.socketRead0(Native Method); java.net.SocketInputStream.read(SocketInputStream.java:129); org.apache.coyote.http11.InternalInputBuffer.fill(InternalInputBuffer.java:767); org.apache.coyote.http11.InternalInputBuffer.parseRequestLine (InternalInputBuffer.java:428); org.apache.coyote.http11.Http11Processor.process (Http11Processor.java:790); org.apache.coyote.http11.Http11Protocol$Http11ConnectionHandler.processConnection (Http11Protocol.java:700); org.apache.tomcat.util.net.TcpWorkerThread.runIt (PoolTcpEndpoint.java:584); org.apache.tomcat.util.threads.ThreadPool$ControlRunnable.run (ThreadPool.java:683); java.lang.Thread.run(Thread.java:619);
	http7077-Processor1	18	RUNNABLE	3.984	0.4528	103	0.014	0	407	562.412	17.27	java.net.SocketInputStream.socketRead0(Native Method); java.net.SocketInputStream.read(SocketInputStream.java:129); org.apache.coyote.http11.InternalInputBuffer.fill(InternalInputBuffer.java:767); org.apache.coyote.http11.InternalInputBuffer.parseRequestLine (InternalInputBuffer.java:428); org.apache.coyote.http11.Http11Processor.process (Http11Processor.java:790);

Figure 3.8: The detailed diagnosis of the CPU utilization of JVM measure

3.3.2 JVM Memory Usage Test

This test monitors every memory type on the JVM and reports how efficiently the JVM utilizes the memory resources of each type.

Note:

- For this test to report detailed diagnostics, the target Java application should use the JDK/JRE offered by one of the following vendors only: Oracle, Sun, OpenJDK , Azul Zing
- If the target Java application is running using an IBM JRE/JDK, then, this test will not report detailed diagnostics. To enable the test to report DD, a MAT plugin is required. Currently, only an eG agent on an AIX system (using an IBM JDF/JRE) can be configured to use this plugin. This plugin needs to be downloaded and extracted into the target AIX host. Once this is done, then the next time the eG agent runs this test, it takes the help of the plugin to read the usage statistics of object types from the heap dump file, and finally reports these metrics to the eG manager. To know how to install and configure the MAT plugin, refer to the Installing and Configuring the MAT Plugin.

Heap dump analysis using the MAT plugin is resource-intensive. It is not recommended for usage in production servers.

- This test can provide detailed diagnosis information for only those monitored Java applications that use **JRE 1.6 or higher**.
- This test can run in an agent-based/agentless manner only, but detailed diagnostics will be available only if the test is run in an agent-based manner.

- For an Azul Zing JVM, you can have this test report additional metrics on heap memory usage by enabling the **MemoryMXBean** interface. MemoryMXBean is an interface used by the Zing management system to access memory-related properties. The MemoryMXBean provides an overview of the memory system and the memory managers that control the size and use patterns of memory. To enable the MemoryMXBean interface, add the following JVM option in the start-up script of the target application.

  -XX:+UseZingMXBeans

  Also, to enable the test to report detailed diagnostics for a Zing JVM, make sure that the following JVM option is included in the start-up script of the target application:

  -XX:+ProfileLiveObjects

Target of the test : A Java application

Agent deploying the test : An internal/remote agent

Outputs of the test : One set of results for every memory type on the JVM being monitored

Configurable parameters for the test

Parameter	Description
Test period	How often should the test be executed
Host	The host for which the test is to be configured.
Port	The port number at which the specified host listens to
Mode	This test can extract metrics from the Java application using either of the following mechanisms: • Using SNMP-based access to the Java runtime MIB statistics; • By contacting the Java runtime (JRE) of the application via JMX To configure the test to use SNMP, select the SNMP option. On the other hand, choose the JMX option to configure the test to use JMX instead. By default, the JMX option is chosen here.
JMX remote port	This parameter appers only if the mode is set to jmx. Here, specify the port at which the jmx listens for requests from remote hosts. Ensure that you specify the same port that you configured in the management.properties file in the <JAVA_

Parameter	Description
	HOME>\jre\lib\management folder used by the target application (see page **3**).
JNDIname	This parameter appears only if the mode is set to JMX. The JNDIname is a lookup name for connecting to the JMX connector. By default, this is jmxrmi. If you have resgistered the JMX connector in the RMI registery using a different lookup name, then you can change this default value to reflect the same.
User, Password, and Confirm password	These parameters appear only if the Mode is set to JMX. If JMX requires authentication only (but no security), then ensure that the user and password parameters are configured with the credentials of a user with read-write access to JMX. To know how to create this user, refer to Section **2.1.2**. Confirm the password by retyping it in the confirm password text box.
Provider	This parameter appears only if the Mode is set to JMX. This test uses a JMX Provider to access the MBean attributes of the target Java application and collect metrics. Specify the package name of this JMX Provider here. By default, this is set to com.sun.jmx.remote.protocol.
Timeout	Specify the duration (in seconds) for which this test should wait for a response from the target Java application. If there is no response from the target beyond the configured duration, the test will timeout. By default, this is set to 240 seconds if the Mode is JMX, and 10 seconds if the Mode is SNMP.
SNMPPort	This parameter appears only if the Mode is set to SNMP. Here specify the port number through which the server exposes its SNMP MIB. Ensure that you specify the same port you configured in the management.properties file in the <JAVA_ HOME>\jre\lib\management folder used by the target application (see page **18**).
SNMP Version	This parameter appears only if the Mode is set to SNMP. The default selection in the SNMP version list is v1. However, for this test to work, you have to select SNMP v2 or v3 from this list, depending upon which version of SNMP is in use in the target environment.
SNMP Community	This parameter appears only if the Mode is set to SNMP. Here, specify the SNMP community name that the test uses to communicate with the mail server. The default is public. This parameter is specific to SNMP v1 and v2 only. Therefore, if the SNMP version chosen is v3, then this parameter will not appear.
User name	This parameter appears only when v3 is selected as the SNMP version. SNMP version 3 (SNMPv3) is an extensible SNMP Framework which supplements the SNMPv2 Framework, by additionally supporting message security, access control, and remote SNMP configuration capabilities. To extract performance statistics from the MIB using the highly secure SNMP v3 protocol, the eG agent has to be configured with the required access privileges – in other words, the eG agent should connect to the MIB

Parameter	Description
	using the credentials of a user with access permissions to be MIB. Therefore, specify the name of such a user against this parameter.
Authpass	Specify the password that corresponds to the above-mentioned user name. This parameter once again appears only if the snmpversion selected is v3.
Confirm password	Confirm the Authpass by retyping it here
Authtype	This parameter too appears only if v3 is selected as the snmpversion. From the authtype list box, choose the authentication algorithm using which SNMP v3 converts the specified username and password into a 32-bit format to ensure security of SNMP transactions. You can choose between the following options: • MD5 – Message Digest Algorithm • SHA – Secure Hash Algorithm
Encryptflag	This flag appears only when v3 is selected as the snmpversion. By default, the eG agent does not encrypt SNMP requests. Accordingly, the flag is set to No by default. To ensure that SNMP requests sent by the eG agent are encrypted, select the Yes option.
Encrypttype	If the Encryptflag is set to Yes, then you will have to mention the encryption type by selecting an option from the Encrypttype list. SNMP v3 supports the following encryption types: • DES – Data Encryption Standard • AES – Advanced Encryption Standard
Encryptpassword	Specify the encryption password here.
Confirm password	Confirm the encryption password by retyping it here.
Data over TCP	This parameter is applicable only if mode is set to SNMP. By default, in an IT environment, all data transmission occurs over UDP. Some environments however, may be specifically configured to offload a fraction of the data traffic – for instance, certain types of data traffic or traffic pertaining to specific components – to other protocols like TCP, so as to prevent UDP overloads. In such environments, you can instruct the eG agent to conduct the SNMP data traffic related to the monitored target over TCP (and not UDP). For this, set this flag to Yes. By default, this flag is set to No.
Heap Analysis	By default, this flag is set to off. This implies that the test will not provide detailed diagnosis information for memory usage, by default. To trigger the collection of detailed measures, set this flag to On.

Parameter	Description
	Note: • If heap analysis is switched On, then the eG agent will be able to collect detailed measures only if the Java application being monitored uses JDK 1.6 or higher. • Heap analytics / detailed diagnostics will be provided only if the Java application being monitored supports Oracle Hotspot.
Java Home	This parameter appears only when the Heap Analysis flag is switched On. Here, provide the full path to the install directory of JDK 1.6 or higher on the application host. For example, *c:\JDK1.6.0*.
Exclude Packages	The detailed diagnosis of this test, if enabled, lists the Java classes/packages that are using the pool memory and the amount of memory used by each class/package. To enable administrators to focus on the memory consumed by those classes/packages that are specific to their application, without being distracted by the memory consumption of basic Java classes/packages, the test, by default, excludes some common Java packages from the detailed diagnosis. The packages excluded by default are as follows: • All packages that start with the string java or javax - in other words, java.* and javax.*. • Arrays of primitive data types - eg., [Z, which is a one-dimensional array of type boolean, [[B, which is a 2-dimensional array of type byte, etc. • A few class loaders - eg., <symbolKlass>, <constantPoolKlass>, <instanceKlassKlass>, <constantPoolCacheKlass>, etc. This is why, the Exclude Packages parameter is by default configured with the packages mentioned above. You can, if required, append more packages or patterns of packages to this comma-separated list. This will ensure that such packages also are excluded from the detailed diagnosis of the test. Note that the exclude packages parameter is of relevance only if the Heap Analysis flag is set to 'Yes'.
Include Packages	By default, this is set to all. This indicates that, by default, the detailed diagnosis of the test (if enabled) includes all classes/packages associated with the monitored Java application, regardless of whether they are basic Java packages or those that are crucial to the functioning of the application. However, if you want the detailed diagnosis to provide the details of memory consumed by a specific set of classes/packages alone, then, provide a comma-separated list of classes/packages to be included in the detailed diagnosis in the include packages text box. Note that the include packages parameter is of relevance only if the Heap Analysis flag is set to 'Yes'.

Parameter	Description
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 this parameter.
Detailed Diagnosis	To make diagnosis more efficient and accurate, the eG Enterprise suite embeds an optional detailed diagnostic capability. With this capability, the eG agents can be configured to run detailed, more elaborate tests as and when specific problems are detected. To enable the detailed diagnosis capability of this test for a particular server, choose the **On** option. To disable the capability, click on the **Off** option. The option to selectively enable/disable the detailed diagnosis capability will be available only if the following conditions are fulfilled: • The eG manager license should allow the detailed diagnosis capability • Both the normal and abnormal frequencies configured for the detailed diagnosis measures should not be 0.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Initial memory	Indicates the amount of memory initially allocated at startup.	MB	
Used memory	Indicates the amount of memory currently used.	MB	It includes the memory occupied by all objects, including both reachable and unreachable objects. Ideally, the value of this measure should be low. A high value or a consistent increase in the value could indicate gradual erosion of memory resources. In such a situation, you can take the help of the detailed diagnosis of this measure (if enabled), to figure out which class is using up memory excessively.
Committed memory	Indicates the amount of memory	MB	The amount of Committed memory may change over time. The Java virtual machine may release

Measurement	Description	Measurement Unit	Interpretation
	guaranteed to be available for use by the JVM.		memory to the system and committed memory could be less than the amount of memory initially allocated at startup. Committed will always be greater than or equal to used memory.
Free memory	Indicates the amount of memory currently available for use by the JVM.	MB	If an Azul Zing JVM is being monitored, then the value of this measure will change according to the value of the *Memory pool size type* measure. If the Memory pool size type measure is Fixed, then the value of this measure is the difference between the value of the *Max allocated memory* and *Used memory* measures. On the other hand, if the *Memory pool size type* is Elastic, then the value of this measure will be the difference between the value of the *Committed memory* and *Used memory m*easures. For all other JVMs, this measure is the difference between the *Max allocated memory* and *Used memor*y measures. Ideally, the value of this measure should be high. **Note:** Sometimes, administrators may not want to cap/limit the maximum amount of memory that a JVM can use. In such cases, they may set the maximum memory to -1. If this is done, then it implies that the JVM can use any amount of memory. In this case therefore, the Maximum allocated memory will also report the value -1, but the Free memory measure will not be reported.
Max allocated memory	Indicates the maximum amount of memory allocated for the JVM.	MB	In the case of the Azul Zing JVM, this measure will be reported only for memory pools of type FIXED.
Used percentage	Indicates the percentage of used memory.	Percent	In the case of the Azul Zing JVM, this measure will be reported only for memory pools of type FIXED. The formula for computing the value of this

<table>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
<tr><td></td><td></td><td></td><td>measure for a FIXED memory type is as follows:

(Used memory / Committed memory)*100

For all other JVMs, the value of this measure is computed using the following formula:

(Used memory / Max allocated memory)*100

Ideally, the value of this measure should be low. A very high value of this measure could indicate excessive memory consumption by the JVM, which in turn, could warrant further investigation. In such a situation, you can take the help of the detailed diagnosis of this measure (if enabled), to figure out which class is using up memory excessively.</td></tr>
<tr><td>JVM heap memory exceeds initially reserved?</td><td>Indicates whether/not the heap memory usage has exceeded the amount of memory initially reserved for this memory type.</td><td></td><td>The values that this measure can report and their corresponding numeric values are listed in the table below:

<table><tr><th>Measure Value</th><th>Numeric Value</th></tr><tr><td>Yes</td><td>1</td></tr><tr><td>No</td><td>0</td></tr></table>
Typically, the value of this measure will be Yes for a memory pool of type ELASTIC - i.e., for those descriptors that report the value ELASTIC for the Memory pool size type measure. For FIXED memory pools on the other hand, the value of this measure will generally be No.

Note:

By default, the test reports the Measure Values listed in the table above to indicate whether/not memory usage has exceeded allocation. In the graph of this measure however, the same is indicated using the numeric equivalents only.</td></tr>
<tr><td>Percentage heap used after GC</td><td>Indicates the percentage of heap</td><td>Percent</td><td>This measure is reported only for Azul Zing JVM, and only when the 'MemoryMXBean'</td></tr>
</table>

Measurement	Description	Measurement Unit	Interpretation
	memory used by the Zing JVM after garbage collection.		**interface is enabled.** **This measure is only reported for the 'Heap memory usage' descriptor.** If garbage collection reclaims a large amount of memory from the Zing JVM, then the value of this measure will be low. On the other hand, if the garbage collector does not reclaim much memory, or if the Java application suddenly runs a memory-intensive process when GC is being performed, then the value of this measure will be very high.
Heap occupied by application objects	Indicates the amount of heap memory that is used by application objects.	MB	**This measure is reported only for Azul Zing JVM, and only when the 'MemoryMXBean' interface is enabled.** **This measure is only reported for the 'Heap memory usage' descriptor.**
Heap reserved for holding application objects	Indicates the amount of heap memory that was reserved for the usage of application objects.	MB	**This measure is reported only for Azul Zing JVM, and only when the 'MemoryMXBean' interface is enabled.** **This measure is only reported for the 'Heap memory usage' descriptor.**
Percentage of heap used by application objects	Indicates what percentage of its reserved memory the application objects used.	Percent	**This measure is reported only for Azul Zing JVM, and only when the 'MemoryMXBean' interface is enabled.** **This measure is only reported for the 'Heap memory usage' descriptor.** The formula used for computing the value of this measure is as follows: *(Heap occupied by application objects/Heap reserved for holding application objects)*100* A value close to 100% is a cause for concern as it indicates that space reserved for application objects is being eroded rapidly. If the space-drain is

<table>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
<tr><td></td><td></td><td></td><td>not controlled, it can cause application performance to deteriorate.</td></tr>
<tr><td>Memory pool size type</td><td>Indicates this memory pool's type in terms of size</td><td></td><td>This measure is reported only for Azul Zing JVM, and only when the 'MemoryMXBean' interface is enabled.
The values that this measure can report and their numeric values are listed in the table below:
<table><tr><th>Measure Value</th><th>Numeric Value</th></tr><tr><td>Uninitialized</td><td>0</td></tr><tr><td>Fixed</td><td>1</td></tr><tr><td>Elastic</td><td>2</td></tr></table>
Each of these measure values are described below:
<table><tr><th>Measure Value</th><th>Description</th></tr><tr><td>Uninitialized</td><td>A memory pool may be labelled as Unitialized, if the memory region itself has not been reserved.</td></tr><tr><td>Fixed</td><td>A Fixed pool has a fixed upper bound on its size, usually set when the memory for that memory pool is initially reserved.</td></tr><tr><td>Elastic</td><td>An Elastic memory pool has the ability to use unused memory that other memory pools are not using. Likewise, when the memory pool no longer needs to use all of its memory, it can return that memory so that it can be used by other sibling memory pools.</td></tr></table>
Note:
By default, the test reports the Measure Values</td></tr>
</table>

Measurement	Description	Measurement Unit	Interpretation
			listed in the table above to indicate the memory pool size type. In the graph of this measure however, the same is indicated using the numeric equivalents only.

The detailed diagnosis of the Used memory measure, if enabled, lists all the classes that are using the pool memory, the amount and percentage of memory used by each class, the number of instances of each class that is currently operational, and also the percentage of currently running instances of each class. Since this list is by default sorted in the descending order of the percentage memory usage, the first class in the list will obviously be the leading memory consumer.

Details of JVM Heap Usage

Time	Class Name	Instance Count	Instance Percentage	Memory used(MB)	Percentage memory used
Jun 17, 2009 12:11:01					
	com.ibc.object.SapBusinessObject	104003	11.5774	12.629	22.5521
	[Ljava.lang.Object;	23586	2.6255	7.4904	13.3759
	<constMethodKlas>	41243	4.5911	5.8357	10.4211
	java.lang.String	174044	19.3742	3.9836	7.1136
	[C	240000	26.7163	3.6621	6.5396
	[B	7336	0.8166	3.5868	6.4051
	<methodKlas>	41243	4.5911	3.1514	5.6275
	<symbolKlas>	69152	7.6979	2.8014	5.0025
	[I	26240	2.921	2.3018	4.1105
	<constantPoolKlas>	3097	0.3448	2.0491	3.6591
	<instanceKlassKlas>	3097	0.3448	1.296	2.3144
	<constantPoolCacheKlas>	2663	0.2964	1.2536	2.2386
	[S	5546	0.6174	0.4283	0.7649
	java.util.Hashtable$Entry	15908	1.7708	0.3641	0.6502
	<methodDataKlas>	870	0.0968	0.3594	0.6418
	java.lang.reflect.Method	4269	0.4752	0.3257	0.5816
	java.lang.Class	3383	0.3766	0.3097	0.5531
	java.util.Vector	13266	1.4767	0.3036	0.5422

Figure 3.9: The detailed diagnosis of the Used memory measure

3.3.2.1 Installing and Configuring the MAT Plugin

Before installing the MAT plugin, make sure that the following requirements are in place:

- To make optimum use of the available memory, the eG agent on AIX runs the *JVM Memory Usage* test as a separate process. Sufficient memory should be available to this process to analyze heap dump. This memory size depends upon heap dump size. For instance, to analyse heap dump of size 2 GB, the process needs free memory of size 4 GB. This implies that the process should be sized with 100% more memory than the heap dump size. This can be

configured in the following manner:

- Edit the eg_tests.ini file in the /opt/egurkha/manager/config directory
- In the [DD_ROWS] section of the file, configure the following parameters:

  Xms = <Initial heap size>

  Xmx=<Maximum heap size>

  By default, both these parameters are set to 2048M. You can change the value of these parameters based on what the heap dump size is.
- Finally, save the file.

- Next, make sure that the /tmp directory in the AIX system has sufficient free space. This is because, the heap dump files will be stored and analyzed in this folder only. The space requirement of this folder too will vary with heap dump size. For instance, if the heap dump size is 40 MB, then for the MAT plugin to analyze memory usage, 50% more free space is required in the /tmp directory – i.e., 60 MB of free space.

To install the plugin, do the following:

1. Download the MAT plugin and extract it into the *egurkha* directory on the AIX system hosting the eG agent. The steps in this regard are as follows:
   - Download the MAT plugin, **mat_plugin_AIX_PPC64.tar.gz**, from the site.
   - Copy the plugin to the /opt/egurkha folder.
   - Next, issue the following command from the AIX shell prompt, to unzip the MAT plugin zip file:

     gunzip mat_plugin_AIX_PPC64.tar.gz
   - Next, issue the following command:

     tar –xvf mat_plugin_AIX_PPC64.tar
   - A folder named 'mat' will now be created in the /opt/egurkha folder.
2. Configure the Java application/container on AIX to perform heap analysis. For this, insert the following in the JVM arguments section of the start-up script of the application/application server, and then save the file:

- Xdump:java:none - Xdump:heap:file=/tmp/egi_heapdump.%Y%m%d.%H%M%S.%pid.%seq.phd

3. You may also want to fine-tune the following parameters in the [DD_ROWS] section of the eg_tests.ini file:

   - **Heap_File_Generation_TimeOut** – (Default: 5 minutes) This is the time up to which the eG agent can write to the heap dump file. If the agent tries to continue writing to the file after this duration, then the file will be deleted. This means, that no DD will be reported by the agent. To avoid this, set the timeout period according to the heap dump size. Larger the size of the heap dump, higher should be the timeout value.
   - **Heap_Analyzing_TimeOut** – (Default: 120 minutes) This is the time up to which the MAT plugin should read and analyze the data in the heap dump file . If the plugin is not able to finish analyzing the data in the heap dump file within the configured duration, then detailed diagnostics will not be reported. To avoid this, set the time period according to the heap dump size. Larger the size of the heap dump, higher should be the timeout value.

4. Finally, restart the application / application server.

3.3.3 JVM Uptime Test

This test tracks the uptime of a JVM. Using information provided by this test, administrators can determine whether the JVM was restarted. Comparing uptime across Java applications, an admin can determine the JVMs that have been running without any restarts for the longest time.

Target of the test : A Java application

Agent deploying the test : An internal/remote agent

Outputs of the test : One set of results for every Java application monitored

Configurable parameters for the test

Parameter	Description
Test period	How often should the test be executed
Host	The host for which the test is to be configured.
Port	The port number at which the specified host listens to
Mode	This test can extract metrics from the Java application using either of the following mechanisms:

Parameter	Description
	• Using SNMP-based access to the Java runtime MIB statistics; • By contacting the Java runtime (JRE) of the application via JMX To configure the test to use SNMP, select the SNMP option. On the other hand, choose the JMX option to configure the test to use JMX instead. By default, the JMX option is chosen here.
JMX remote port	This parameter appers only if the mode is set to jmx. Here, specify the port at which the jmx listens for requests from remote hosts. Ensure that you specify the same port that you configured in the management.properties file in the <JAVA_HOME>\jre\lib\management folder used by the target application (see page **3**).
JNDIname	This parameter appears only if the mode is set to JMX. The JNDIname is a lookup name for connecting to the JMX connector. By default, this is jmxrmi. If you have resgistered the JMX connector in the RMI registery using a different lookup name, then you can change this default value to reflect the same.
User, Password, and Confirm password	These parameters appear only if the Mode is set to JMX. If JMX requires authentication only (but no security), then ensure that the user and password parameters are configured with the credentials of a user with read-write access to JMX. To know how to create this user, refer to Section **2.1.2**. Confirm the password by retyping it in the confirm password text box.
Provider	This parameter appears only if the Mode is set to JMX. This test uses a JMX Provider to access the MBean attributes of the target Java application and collect metrics. Specify the package name of this JMX Provider here. By default, this is set to com.sun.jmx.remote.protocol.
Timeout	Specify the duration (in seconds) for which this test should wait for a response from the target Java application. If there is no response from the target beyond the configured duration, the test will timeout. By default, this is set to 240 seconds if the Mode is JMX, and 10 seconds if the Mode is SNMP.
SNMPPort	This parameter appears only if the Mode is set to SNMP. Here specify the port number through which the server exposes its SNMP MIB. Ensure that you specify the same port you configured in the management.properties file in the <JAVA_HOME>\jre\lib\management folder used by the target application (see page **18**).
SNMP Version	This parameter appears only if the Mode is set to SNMP. The default selection in the SNMP version list is v1. However, for this test to work, you have to select SNMP v2 or v3 from this list, depending upon which version of SNMP is in use in the target environment.

Parameter	Description
SNMP Community	This parameter appears only if the Mode is set to SNMP. Here, specify the SNMP community name that the test uses to communicate with the mail server. The default is public. This parameter is specific to SNMP v1 and v2 only. Therefore, if the SNMP version chosen is v3, then this parameter will not appear.
User name	This parameter appears only when v3 is selected as the SNMP version. SNMP version 3 (SNMPv3) is an extensible SNMP Framework which supplements the SNMPv2 Framework, by additionally supporting message security, access control, and remote SNMP configuration capabilities. To extract performance statistics from the MIB using the highly secure SNMP v3 protocol, the eG agent has to be configured with the required access privileges – in other words, the eG agent should connect to the MIB using the credentials of a user with access permissions to be MIB. Therefore, specify the name of such a user against this parameter.
Context	This parameter appears only when v3 is selected as the SNMPVERSION. An SNMP context is a collection of management information accessible by an SNMP entity. An item of management information may exist in more than one context and an SNMP entity potentially has access to many contexts. A context is identified by the SNMPEngineID value of the entity hosting the management information (also called a contextEngineID) and a context name that identifies the specific context (also called a contextName). If the USERNAME provided is associated with a context name, then the eG agent will be able to poll the MIB and collect metrics only if it is configured with the context name as well. In such cases therefore, specify the context name of the username in the context text box. By default, this parameter is set to none.
Authpass	Specify the password that corresponds to the above-mentioned user name. This parameter once again appears only if the snmpversion selected is v3.
Confirm password	Confirm the Authpass by retyping it here
Authtype	This parameter too appears only if v3 is selected as the snmpversion. From the authtype list box, choose the authentication algorithm using which SNMP v3 converts the specified username and password into a 32-bit format to ensure security of SNMP transactions. You can choose between the following options: • MD5 – Message Digest Algorithm • SHA – Secure Hash Algorithm
Encryptflag	This flag appears only when v3 is selected as the snmpversion. By default, the eG agent does not encrypt SNMP requests. Accordingly, the flag is set to No by default. To ensure that SNMP requests sent by the eG agent are encrypted, select the Yes option.

Parameter	Description
Encrypttype	If the Encryptflag is set to Yes, then you will have to mention the encryption type by selecting an option from the Encrypttype list. SNMP v3 supports the following encryption types: • DES – Data Encryption Standard • AES – Advanced Encryption Standard
Encryptpassword	Specify the encryption password here.
Confirm password	Confirm the encryption password by retyping it here.
Data over TCP	This parameter is applicable only if mode is set to SNMP. By default, in an IT environment, all data transmission occurs over UDP. Some environments however, may be specifically configured to offload a fraction of the data traffic – for instance, certain types of data traffic or traffic pertaining to specific components – to other protocols like TCP, so as to prevent UDP overloads. In such environments, you can instruct the eG agent to conduct the SNMP data traffic related to the monitored target over TCP (and not UDP). For this, set this flag to Yes. By default, this flag is set to No.
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 this parameter.
Detailed Diagnosis	To make diagnosis more efficient and accurate, the eG Enterprise suite embeds an optional detailed diagnostic capability. With this capability, the eG agents can be configured to run detailed, more elaborate tests as and when specific problems are detected. To enable the detailed diagnosis capability of this test for a particular server, choose the On option. To disable the capability, click on the Off option. The option to selectively enable/disable the detailed diagnosis capability will be available only if the following conditions are fulfilled: • The eG manager license should allow the detailed diagnosis capability • Both the normal and abnormal frequencies configured for the detailed diagnosis measures should not be 0.

Measurements made by the test

<table>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
<tr><td>Has JVM been restarted?</td><td>Indicates whether or not the JVM has restarted during the last measurement period.</td><td></td><td>If the value of this measure is No, it indicates that the JVM has not restarted. The value Yes on the other hand implies that the JVM has indeed restarted.
The numeric values that correspond to the restart states discussed above are listed in the table below:
<table><tr><th>State</th><th>Value</th></tr><tr><td>Yes</td><td>1</td></tr><tr><td>No</td><td>0</td></tr></table>
Note:
By default, this measure reports the value Yes or No to indicate whether a JVM has restarted. The graph of this measure however, represents the same using the numeric equivalents – 0 or 1.</td></tr>
<tr><td>Uptime during the last measure period</td><td>Indicates the time period that the JVM has been up since the last time this test ran.</td><td>Secs</td><td>If the JVM has not been restarted during the last measurement period and the agent has been running continuously, this value will be equal to the measurement period. If the JVM was restarted during the last measurement period, this value will be less than the measurement period of the test. For example, if the measurement period is 300 secs, and if the JVM was restarted 120</td></tr>
</table>

Measurement	Description	Measurement Unit	Interpretation
			secs back, this metric will report a value of 120 seconds. The accuracy of this metric is dependent on the measurement period – the smaller the measurement period, greater the accuracy.
Total Uptime of the JVM	Indicates the total time that the JVM has been up since its last reboot.	Secs	Administrators may wish to be alerted if a JVM has been running without a reboot for a very long period. Setting a threshold for this metric allows administrators to determine such conditions.

3.3.4 JVM Leak Suspects Test

Note:

This test is CPU & Memory intensive and can cause slowness to the underlying application. It is hence NOT advisable to enable this test on production environments. It is ideally suited for Development and Staging environments.

Java implements automatic garbage collection (GC); once you stop using an object, you can depend on the garbage collector to collect it. To stop using an object, you need to eliminate all references to it. However, when a program never stops using an object by keeping a permanent reference to it, memory leaks occur. For example, let's consider the piece of code below:

```java
import java.util.ArrayList;
import java.util.List;

class MemoryLeakDemo {
    private static List<Integer> memoryLeakArea = new ArrayList<Integer>();

    public static void main(String [] args) {

        int iteration = 0;

        // This infinite loop would eventually run out of memory
        while (true) {
            // Add number of the current iteration to the list.
            Integer payload = new Integer(iteration);
            memoryLeakArea.add(payload);
            iteration++;
        }
    }
}
```

Figure 3.10: A sample code

In the example above, we continue adding new elements to the list memoryLeakArea without ever removing them. In addition, we keep references to the memoryLeakArea, thereby preventing GC from collecting the list itself. So although there is GC available, it cannot help because we are still using memory. The more time passes the more memory we use, which in effect requires an infinite amount memory for this program to continue running. When no more memory is remaining, an OutOfMemoryError alert will be thrown and generate an exception like this: Exception in thread "main" java.lang.OutOfMemoryError: Java heap space at MemoryLeakDemo.main (MemoryLeakDemo.java:14)

Typically, such alerts signal a potential memory leak!

A memory leak can diminish the performance of your mission-critical Java applications by reducing the amount of available memory. Eventually, in the worst case, it may cause the application to crash due to thrashing. To avert such unwarranted application failures, it is imperative that memory leaks are detected at the earliest and the objects responsible for them accurately isolated. This is where, the **JVM Leak Suspects** test helps! This test continuously monitors the JVM heap usage and promptly alerts administrators when memory usage crosses a configured limit. The detailed diagnostics of the test will then lead you to the classes that are consuming memory excessively, thereby pointing you to those classes that may have caused the leak.

Note:

This test will work only if the following pre-requisites are fulfilled:

- The test should be executed in an agent-based manner only.
- The target Java application should use the JDK/JRE offered by one of the following vendors only: Oracle, Sun, OpenJDK. IBM JDK/JRE is not supported.
- The monitored Java application should use JDK/JRE 1.6 or higher.
- For this test to run and report metrics, the eG agent install user should be the same as the Java application (or) Java web/application server install user.
- By default , this test programmatically dumps a heap dump (.hprof files) in the folder <EG_AGENT_INSTALL_DIR>\agent\logs folder. To enable the eG agent to read/analyse such files, you need to add the eG agent install user to the Java application (or) Java web/application server install user group. If this is not done, then the dump files will be created, but will not be processed by the eG agent, thus ending up unnecessarily occupying disk space (note that .hprof files are normally 1-5 GB in size).

This test is disabled by default. To enable the test, go to the **ENABLE / DISABLE TESTS** page using the menu sequence : Agents -> Tests -> Enable/Disable, pick the desired **Component type**, set *Performance* as the **Test type**, choose the test from the **DISABLED TESTS** list, and click on the **<** button to move the test to the **ENABLED TESTS** list. Finally, click the **Update** button.

Target of the test : A Java application

Agent deploying the test : An internal/remote agent

Outputs of the test : One set of results for every Java application monitored

Configurable parameters for the test

Parameter	Description
Test period	How often should the test be executed
Host	The host for which the test is to be configured.
Port	The port number at which the specified host listens to
Mode	This test can extract metrics from the Java application using either of the following mechanisms: • Using SNMP-based access to the Java runtime MIB statistics; • By contacting the Java runtime (JRE) of the application via JMX

Parameter	Description
	To configure the test to use SNMP, select the SNMP option. On the other hand, choose the JMX option to configure the test to use JMX instead. By default, the JMX option is chosen here.
JMX remote port	This parameter appers only if the mode is set to jmx. Here, specify the port at which the jmx listens for requests from remote hosts. Ensure that you specify the same port that you configured in the management.properties file in the <JAVA_HOME>\jre\lib\management folder used by the target application (see page **3**).
JNDIname	This parameter appears only if the mode is set to JMX. The JNDIname is a lookup name for connecting to the JMX connector. By default, this is jmxrmi. If you have resgistered the JMX connector in the RMI registery using a different lookup name, then you can change this default value to reflect the same.
User, Password, and Confirm password	These parameters appear only if the Mode is set to JMX. If JMX requires authentication only (but no security), then ensure that the user and password parameters are configured with the credentials of a user with read-write access to JMX. To know how to create this user, refer to Section **2.1.2**. Confirm the password by retyping it in the confirm password text box.
Provider	This parameter appears only if the Mode is set to JMX. This test uses a JMX Provider to access the MBean attributes of the target Java application and collect metrics. Specify the package name of this JMX Provider here. By default, this is set to com.sun.jmx.remote.protocol.
Timeout	Specify the duration (in seconds) for which this test should wait for a response from the target Java application. If there is no response from the target beyond the configured duration, the test will timeout. By default, this is set to 240 seconds if the Mode is JMX, and 10 seconds if the Mode is SNMP.
SNMPPort	This parameter appears only if the Mode is set to SNMP. Here specify the port number through which the server exposes its SNMP MIB. Ensure that you specify the same port you configured in the management.properties file in the <JAVA_HOME>\jre\lib\management folder used by the target application (see page **18**).
SNMP Version	This parameter appears only if the Mode is set to SNMP. The default selection in the SNMP version list is v1. However, for this test to work, you have to select SNMP v2 or v3 from this list, depending upon which version of SNMP is in use

Parameter	Description
	in the target environment.
SNMP Community	This parameter appears only if the Mode is set to SNMP. Here, specify the SNMP community name that the test uses to communicate with the mail server. The default is public. This parameter is specific to SNMP v1 and v2 only. Therefore, if the SNMP version chosen is v3, then this parameter will not appear.
User name	This parameter appears only when v3 is selected as the SNMP version. SNMP version 3 (SNMPv3) is an extensible SNMP Framework which supplements the SNMPv2 Framework, by additionally supporting message security, access control, and remote SNMP configuration capabilities. To extract performance statistics from the MIB using the highly secure SNMP v3 protocol, the eG agent has to be configured with the required access privileges – in other words, the eG agent should connect to the MIB using the credentials of a user with access permissions to be MIB. Therefore, specify the name of such a user against this parameter.
Context	This parameter appears only when v3 is selected as the SNMPVERSION. An SNMP context is a collection of management information accessible by an SNMP entity. An item of management information may exist in more than one context and an SNMP entity potentially has access to many contexts. A context is identified by the SNMPEngineID value of the entity hosting the management information (also called a contextEngineID) and a context name that identifies the specific context (also called a contextName). If the USERNAME provided is associated with a context name, then the eG agent will be able to poll the MIB and collect metrics only if it is configured with the context name as well. In such cases therefore, specify the context name of the username in the context text box. By default, this parameter is set to none.
Authpass	Specify the password that corresponds to the above-mentioned user name. This parameter once again appears only if the snmpversion selected is v3.
Confirm password	Confirm the Authpass by retyping it here
Authtype	This parameter too appears only if v3 is selected as the snmpversion. From the authtype list box, choose the authentication algorithm using which SNMP v3 converts the specified username and password into a 32-bit format to ensure security of SNMP transactions. You can choose between the following options:

Parameter	Description
	• MD5 – Message Digest Algorithm • SHA – Secure Hash Algorithm
Encryptflag	This flag appears only when v3 is selected as the snmpversion. By default, the eG agent does not encrypt SNMP requests. Accordingly, the flag is set to No by default. To ensure that SNMP requests sent by the eG agent are encrypted, select the Yes option.
Encrypttype	If the Encryptflag is set to Yes, then you will have to mention the encryption type by selecting an option from the Encrypttype list. SNMP v3 supports the following encryption types: • DES – Data Encryption Standard • AES – Advanced Encryption Standard
Encryptpassword	Specify the encryption password here.
Confirm password	Confirm the encryption password by retyping it here.
Data over TCP	This parameter is applicable only if mode is set to SNMP. By default, in an IT environment, all data transmission occurs over UDP. Some environments however, may be specifically configured to offload a fraction of the data traffic – for instance, certain types of data traffic or traffic pertaining to specific components – to other protocols like TCP, so as to prevent UDP overloads. In such environments, you can instruct the eG agent to conduct the SNMP data traffic related to the monitored target over TCP (and not UDP). For this, set this flag to Yes. By default, this flag is set to No.
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 this parameter.
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

Parameter	Description
	this test for a particular server, choose the **On** option. To disable the capability, click on the **Off** option. The option to selectively enable/disable the detailed diagnosis capability will be available only if the following conditions are fulfilled: • The eG manager license should allow the detailed diagnosis capability • Both the normal and abnormal frequencies configured for the detailed diagnosis measures should not be 0.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Allocated Heap Memory	Indicates the total amount of memory space occupied by the objects that are currently loaded on to the JVM.	MB	
Leak suspected classes	Indicates the number of classes that are memory leak suspects.	Number	Use the detailed diagnosis of this measure to know which classes are using more memory than the configured pct heap limit. Remember that all applications/classes that throw OutofMemory exceptions need not be guilty of leaking memory. Such an exception can occur even if a class requires more memory for normal functioning. To distinguish between a memory leak and an application that simply needs more memory, we need to look at the "peak load" concept. When program has just started no users have yet used it, and as a result it typically needs much less memory then when thousands of users are interacting with it. Thus, measuring memory usage immediately after a program starts is not

<table>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
<tr><td></td><td></td><td></td><td>the best way to gauge how much memory it needs! To measure how much memory an application needs, memory size measurements should be taken at the time of peak load—when it is most heavily used. Therefore, it is good practice to check the memory usage of the ‘suspected classes’ at the time of peak load to determine whether they are indeed leaking memory or not.</td></tr>
<tr><td>Number of objects</td><td>Indicates the number of objects present in the JVM.</td><td>Number</td><td>Use the detailed diagnosis of this measure to view the top-20 classes in the JVM in terms of memory usage.</td></tr>
<tr><td>Number of classes</td><td>Indicates the number of classes currently present in the JVM.</td><td>Number</td><td></td></tr>
<tr><td>Number of class loaders</td><td>Indicates the number of class loaders currently present in the JVM.</td><td>Number</td><td></td></tr>
<tr><td>Number of GC roots</td><td>Indicate the number of GC roots currently present in the JVM.</td><td>Number</td><td>A garbage collection root is an object that is accessible from outside the heap. The following reasons make an object a GC root:
<table>
<tr><th>Reason</th><th>Description</th></tr>
<tr><td>System Class</td><td>Class loaded by bootstrap/system class loader. For example, everything from the rt.jar like java.util.</td></tr>
<tr><td>JNI Local</td><td>Local variable in native code, such as user defined JNI</td></tr>
</table>
</td></tr>
</table>

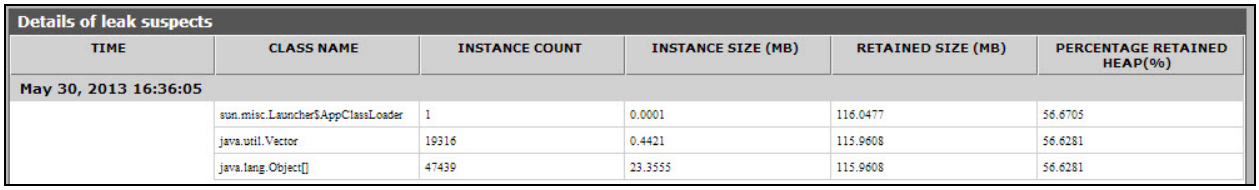
Measurement	Description	Measurement Unit	Interpretation

Reason	Description
	code or JVM internal code
JNI Global	Global variable in native code, such as user defined JNI code or JVM internal code
Thread Block	Object referred to from a currently active thread block
Thread	A started, but not stopped, thread
Busy Monitor	Everything that has called wait() or notify () or that is synchronized. For example, by calling synchronized (Object) or by entering a synchronized method. Static method means class, non-static method means object
Java Local	Local variable. For example, input parameters or locally created objects of methods

<table>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
<tr><td></td><td></td><td></td><td>
<table>
<tr><th>Reason</th><th>Description</th></tr>
<tr><td></td><td>that are still in the stack of a thread.</td></tr>
<tr><td>Native Stack</td><td>In or out parameters in native code, such as user defined JNI code or JVM internal code. or reflection.</td></tr>
<tr><td>Finalizer</td><td>An object which is in a queue awaiting its finalizer to be run.</td></tr>
<tr><td>Unfinalized</td><td>An object which has a finalize method, but has not been finalized and is not yet on the finalizer queue.</td></tr>
<tr><td>Unreachable</td><td>An object which is unreachable from any other root, but has been marked as a root by MAT to retain objects which otherwise would not be included in the analysis.</td></tr>
<tr><td>Unknown</td><td>An object of unknown root type.</td></tr>
</table>
</td></tr>
<tr><td>Objects pending for finalization</td><td>Indicates the number of objects that are pending for finalization.</td><td>Number</td><td>Sometimes an object will need to perform some action when it is destroyed. For example, if an object is holding some non-</td></tr>
</table>

Measurement	Description	Measurement Unit	Interpretation
			java resource such as a file handle or window character font, then you might want to make sure these resources are freed before an object is destroyed. To handle such situations, Java provides a mechanism called finalization. By using finalization, you can define specific actions that will occur when an object is just about to be reclaimed by the garbage collector. A high value for this measure indicates the existence of many objects that are still occupying the JVM memory space and are unable to be reclaimed by GC. A consistent rise in this value is also a sign of a memory leak.

The detailed diagnosis of the *Leak suspected classes* measure lists the names of all classes for which the memory usage is over the configured **PCT HEAP LIMIT.** In addition, the detailed diagnosis also reveals the **PERCENTAGE RETAINED HEAP** of each class - this is the percentage of the total *Allocated heap size* that is used by every class. From this, you can easily infer which class is consuming the maximum memory, and is hence, the key memory leak suspect. By observing the memory usage of this class during times of peak load, you can corroborate eG's findings - i.e., you can know for sure whether that class is indeed leaking memory or not!



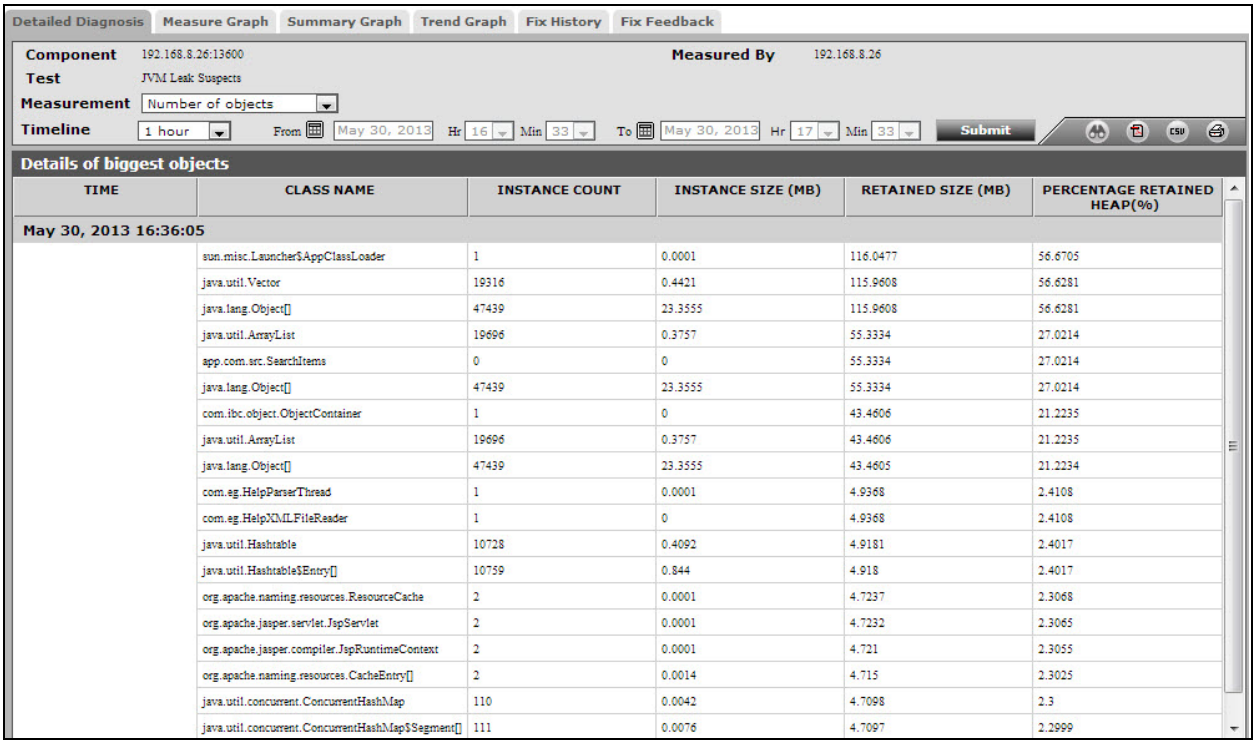
Details of leak suspects

TIME	CLASS NAME	INSTANCE COUNT	INSTANCE SIZE (MB)	RETAINED SIZE (MB)	PERCENTAGE RETAINED HEAP(%)
May 30, 2013 16:36:05					
	sun.misc.Launcher$AppClassLoader	1	0.0001	116.0477	56.6705
	java.util.Vector	19316	0.4421	115.9608	56.6281
	java.lang.Object[]	47439	23.3555	115.9608	56.6281

Figure 3.11: The detailed diagnosis of the Leak suspect classes measure

The detailed diagnosis of the Number of objects measure lists the names of the top-20 classes in the JVM, in terms of memory usage. In addition, the detailed diagnosis also reveals the percentage retained heap of each class - this is the percentage of the total Allocated heap size that is used by every class. From this, you can easily infer which class is consuming the maximum memory, and is hence, the key memory leak suspect. By observing the memory usage of this class during times of

peak load, you can corroborate eG's findings - i.e., you can know for sure whether that class is indeed leaking memory or not!



Detailed Diagnosis | Measure Graph | Summary Graph | Trend Graph | Fix History | Fix Feedback

Component 192.168.8.26:13600 Measured By 192.168.8.26
Test JVM Leak Suspects
Measurement Number of objects
Timeline 1 hour From May 30, 2013 Hr 16 Min 33 To May 30, 2013 Hr 17 Min 33 Submit

Details of biggest objects

TIME	CLASS NAME	INSTANCE COUNT	INSTANCE SIZE (MB)	RETAINED SIZE (MB)	PERCENTAGE RETAINED HEAP(%)
May 30, 2013 16:36:05					
	sun.misc.Launcher$AppClassLoader	1	0.0001	116.0477	56.6705
	java.util.Vector	19316	0.4421	115.9608	56.6281
	java.lang.Object[]	47439	23.3555	115.9608	56.6281
	java.util.ArrayList	19696	0.3757	55.3334	27.0214
	app.com.src.SearchItems	0	0	55.3334	27.0214
	java.lang.Object[]	47439	23.3555	55.3334	27.0214
	com.ibc.object.ObjectContainer	1	0	43.4606	21.2235
	java.util.ArrayList	19696	0.3757	43.4606	21.2235
	java.lang.Object[]	47439	23.3555	43.4605	21.2234
	com.eg.HelpParserThread	1	0.0001	4.9368	2.4108
	com.eg.HelpXMLFileReader	1	0	4.9368	2.4108
	java.util.Hashtable	10728	0.4092	4.9181	2.4017
	java.util.Hashtable$Entry[]	10759	0.844	4.918	2.4017
	org.apache.naming.resources.ResourceCache	2	0.0001	4.7237	2.3068
	org.apache.jasper.servlet.JspServlet	2	0.0001	4.7232	2.3065
	org.apache.jasper.compiler.JspRuntimeContext	2	0.0001	4.721	2.3055
	org.apache.naming.resources.CacheEntry[]	2	0.0014	4.715	2.3025
	java.util.concurrent.ConcurrentHashMap	110	0.0042	4.7098	2.3
	java.util.concurrent.ConcurrentHashMap$Segment[]	111	0.0076	4.7097	2.2999

Figure 3.12: The detailed diagnosis of the Number of objects measure

3.4 What the eG Enterprise Java Monitor Reveals?

This section discusses how administrators can effortlessly and accurately diagnose the root-cause of issues experienced by Java applications, using the class eG JVM Monitor. Each of the sub-sections that follow take the case of a sample application problem, and illustrates the steps to be followed to troubleshoot the problem in the eG monitoring console.

3.4.1 Identifying and Diagnosing a CPU Issue in the JVM

In this section, let us consider the case of the Java application, sapbusiness-152:123, which is being monitored by eG Enterprise. Assume that this application is running on a Tomcat server.

Initially, the application was functioning normally, as indicated by Figure 3.13. There are no high CPU threads.

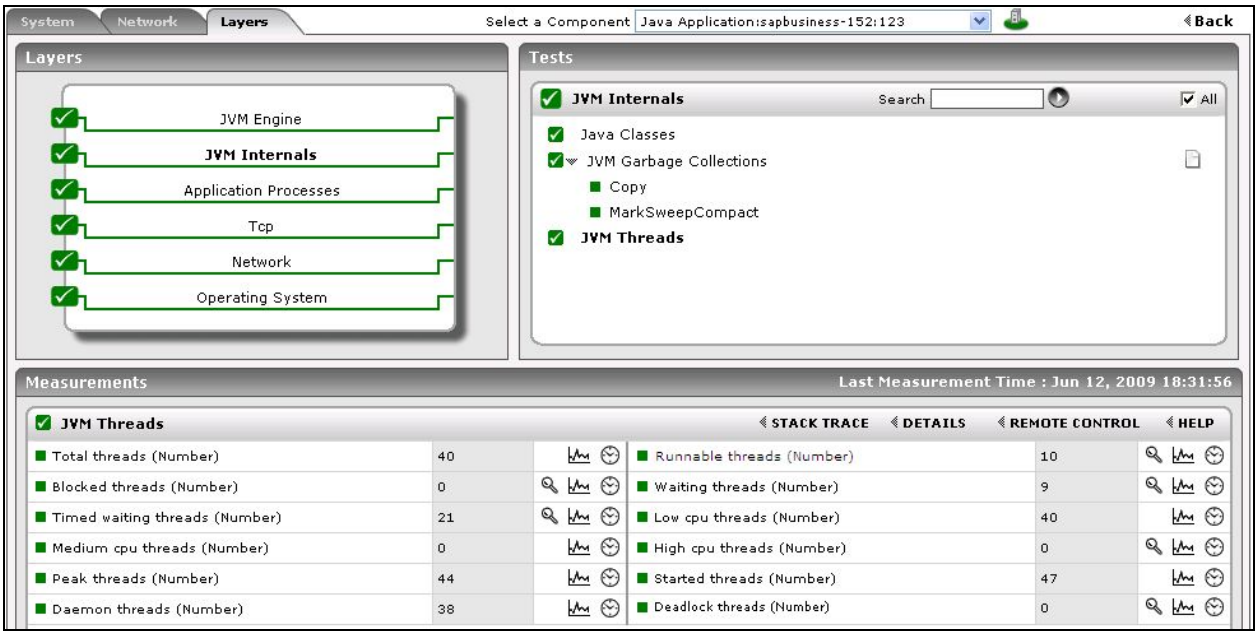


Figure 3.13: The Java application being monitored functioning normally

Now, assume that suddenly, one of the threads executed by the application starts to run abnormally, consuming excessive CPU resources. This is indicated by a change in the value of the High cpu threads measure reported by the jvm Threads test mapped to the jvm Internals layer of the Java Application monitoring model (see Figure 3.13). As you can see, as long as the sapbusiness application was performing well, the value of the High cpu threads measure was 0 (see Figure 3.13). However, as soon as a thread began exhibiting abnormal CPU usage trends, the value changed to 1 (see Figure 3.14).

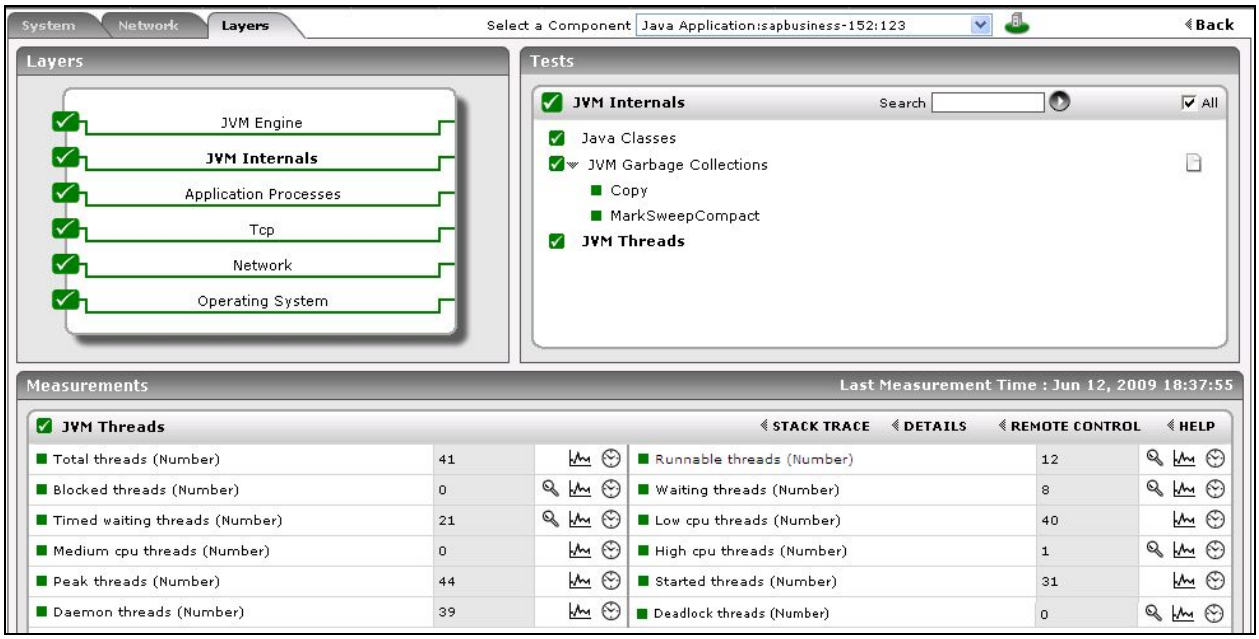


Figure 3.14: The High cpu threads measure indicating that a single thread is consuming CPU excessively

To know which thread is consuming too much CPU, click on the diagnosis icon (i.e., the magnifying glass icon) corresponding to the High cpu threads measure in Figure 3.14. Figure 3.15 then appears revealing the name of the CPU- intensive thread (SapBusinessConnectorThread) and the percentage of CPU used by the thread during the last measurement period. In addition, Figure 3.15 also reveals the number of times the thread was blocked, the total duration of the blocks, the number of times the thread was in waiting, and the percentage of time waited, thereby revealing how resource-intensive the thread has been during the last measurement period.

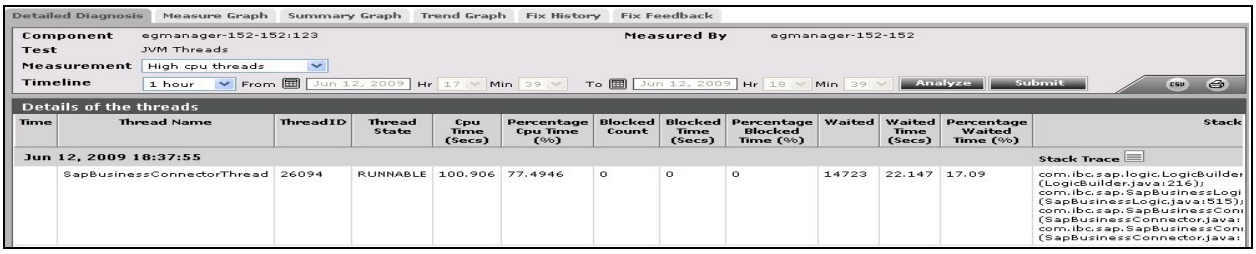


Figure 3.15: The detailed diagnosis of the High cpu threads measure

Let us now get back to the CPU usage issue. Now that we know which thread is causing the CPU usage spike, we next need to determine what is causing this thread to erode the CPU resources. To facilitate this analysis, the detailed diagnosis page of Figure 3.15 also provides the Stack Trace for the thread. You might have to scroll left to view the complete Stack Trace of the thread (see Figure 3.16).

Figure 3.16: Viewing the stack trace as part of the detailed diagnosis of the High cpu threads measure

The stack trace is useful in determining exactly which line of code the thread was executing when we took the last diagnosis snapshot and what was the code execution path that the thread had taken.

To view the stack trace of the CPU-intensive thread more clearly and to analyze it closely, click on the icon in Figure 3.16 or the **Stack Trace** label adjacent to the icon. Figure 3.17 then appears.

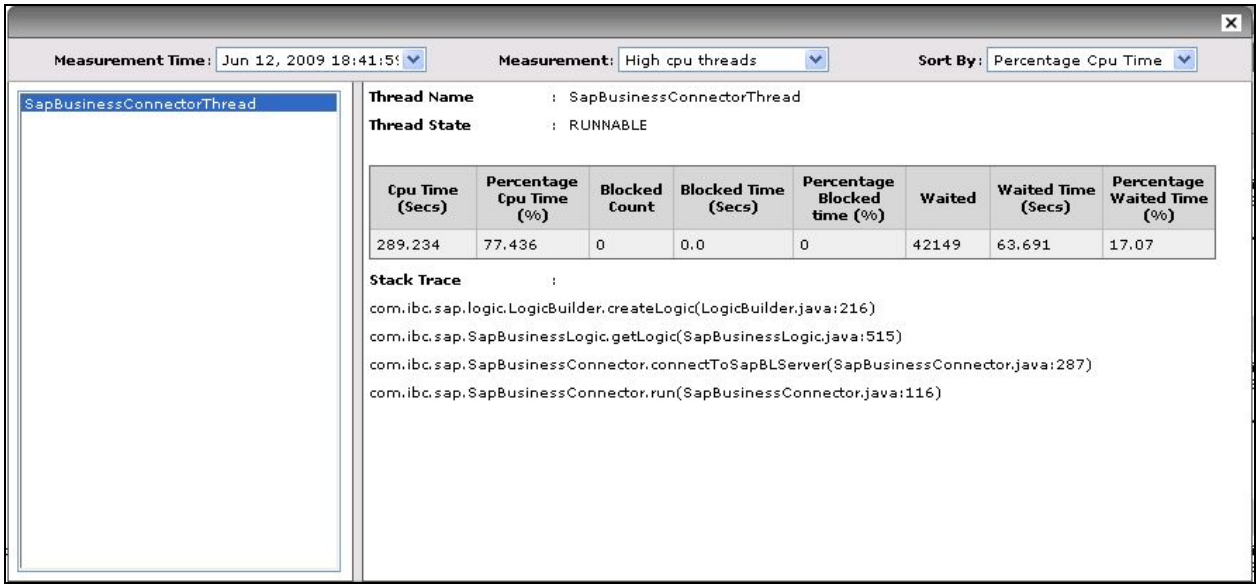


Figure 3.17: Stack trace of the CPU-intensive thread

As you can see, Figure 3.17 provides two panels. The left panel of Figure 3.17, by default, displays all the high CPU-consuming threads sorted in the descending order of their CPU usage. Accordingly, the High cpu threads measure is chosen by default from the Measurement list, and the Percentage Cpu Time is the default selection in the Sort By list in Figure 3.17. These default selections can however be changed by picking a different option from the Measurement and Sort By lists.

The right panel on the other hand, typically displays the current state, overall resource usage, and the Stack Trace for the thread chosen from the left panel. By default however, the right panel provides the stack trace for the leading CPU consumer.

In the case of our example, since only a single thread is currently utilizing CPU excessively, the name of that thread (i.e, SapBusinessConnectorThread) alone will appear in the left panel of Figure 3.17. The right panel too therefore, will display the details of the SapBusinessConnectorThread only. Let us begin to analyze the Stack Trace of this thread carefully.

Stack trace information should always be viewed in a top-down manner. The method most likely to be the cause of the problem is the one on top. In the example of Figure 3.17, this is com.ibc.sap.logic.LogicBuilder.createLogic. The line of code that was executed last when the snapshot was taken is within the createLogic method of the com.ibc.sap.logic.LogicBuilder class. This is line number 216 of the LogicBuilder.java source file. The subsequent lines of the stack trace indicate the sequence of method calls that resulted in com.ibc.sap.logic.LogicBuilder.createLogic being invoked. In this example, com.ibc.sap.logic.LogicBuilder.createLogic has been invoked from the method com.ibc.sap.SapBusinessLogic.getLogic. This invocation has been done by line 515 of SapBusinessLogic.java source file.

To verify if the stack trace is correct in identifying the exact line of the source code that is responsible for the sudden increase in CPU consumption by the SapBusinessConnectorThread, let us review the LogicBuilder.java file in an editor (see Figure 3.18).

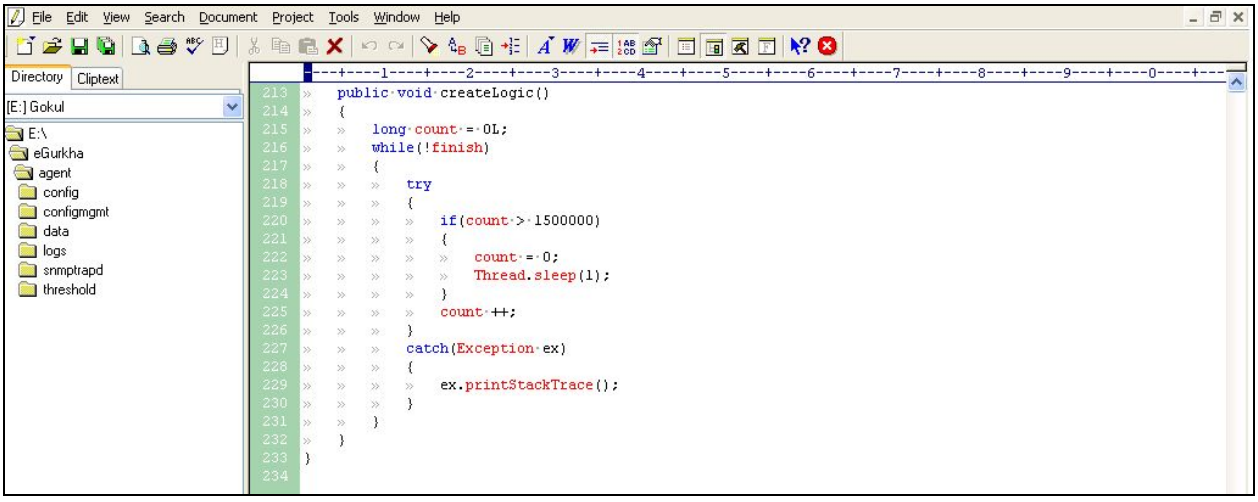


Figure 3.18: The LogicBuilder.java file

Figure 3.18 indicates line 216 of the LogicBuilder.java file. At this line, a while loop seems to have been initiated. This code is supposed to loop 1,500,000 times and then sleep waiting for count to decrease. Instead, a problem in the code – the value of count being reset to 0 at line 222 - is causing

the while loop to execute forever, thereby resulting in one of the threads in the JVM taking a lot of CPU. Deleting the code at line 222 would solve this problem. Once this is done, then the SapConnectorThread will no longer consume too much CPU; this in turn will decrement the value of the High Cpu threads measure by 1 (see Figure 3.19).

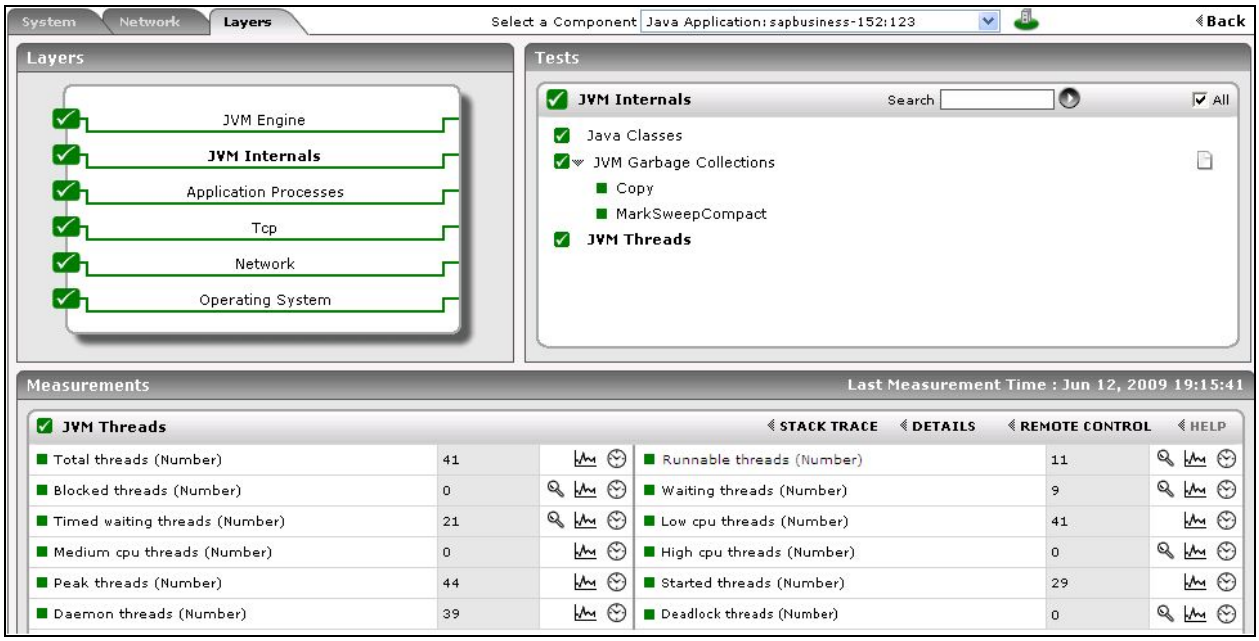


Figure 3.19: The High CPU threads measure reporting a 0 value

With that, we have seen how a simple sequence of steps bundled into the eG JVM Monitor, help an administrator in identifying not only a CPU-intensive thread, but also the exact line of code executed by that thread that could be triggering the spike in usage.

3.4.2 Identifying and Diagnosing a Thread Blocking Issue in the JVM

This section once again takes the example of the sapbusiness application used by Section **3.4.1**. Here, we will see how the eG JVM Monitor instantly identifies blocked threads, and intelligently diagnoses the reason for the blockage.

If a thread executing within the sapbusiness application gets blocked, the value of the Blocked threads measure reported by the jvm Threads test mapped to the **JVM Internals** layer, gets incremented by 1. When this happens, eG Enterprise automatically raises this as a problem condition and changes the state of the Blocked threads measure (see Figure 3.20).

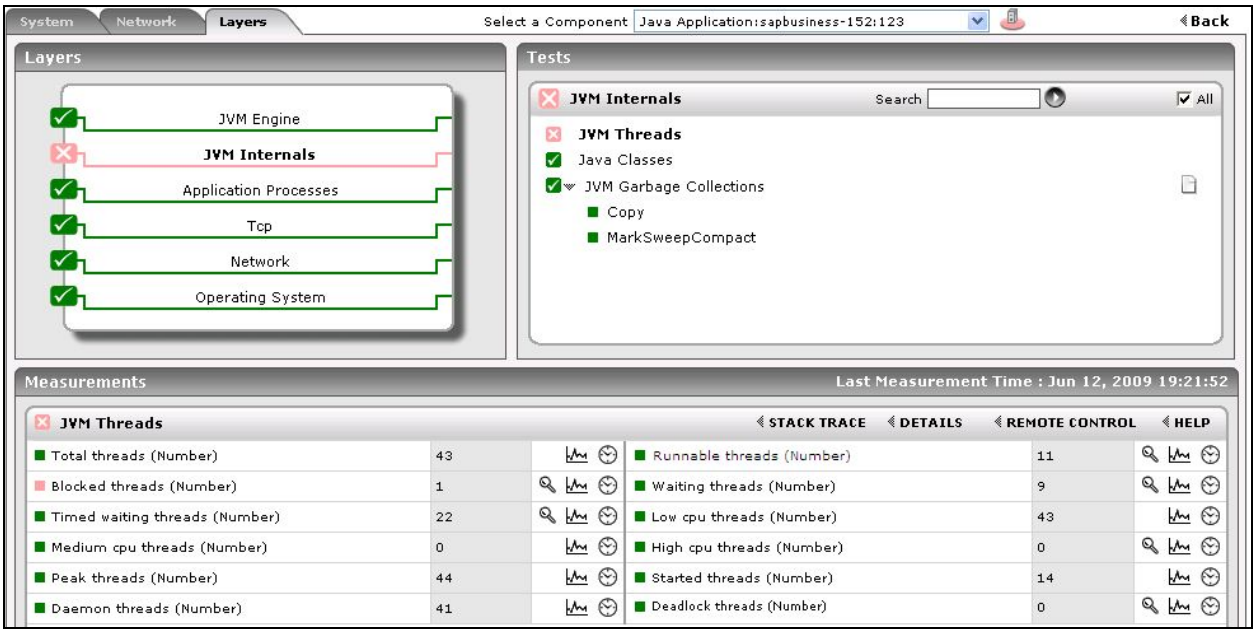


Figure 3.20: The value of the Blocked threads measure being incremented by 1

According to Figure 3.20, the eG JVM Monitor has detected that a thread running in the sapclient application has been blocked. To know which thread this is and for how long it has been blocked, click on the **DIAGNOSIS** icon corresponding to the *Blocked threads* measure. Figure 3.21 will then appear revealing the name of the blocked thread, how long it was blocked, the CPU usage of the thread, and the time for which the thread was in waiting.

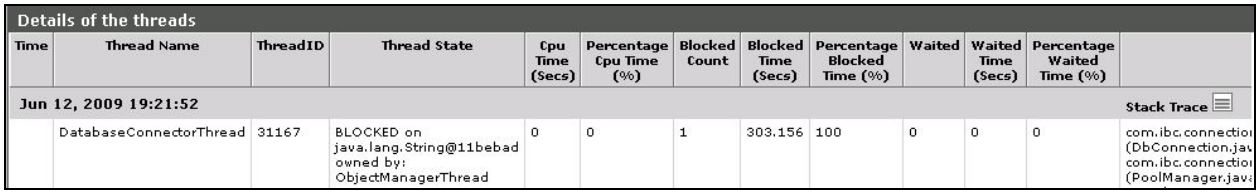


Figure 3.21: Figure 52: The detailed diagnosis of the Blocked threads measure revealing the details of the blocked thread

Figure 3.21 clearly indicates that the DatabaseConnectorThread running in the sapbusiness application was blocked 100% of the time. The next step is to figure out who or what is blocking the thread, and why. To achieve this, we need to analyze the stack trace information of the blocked thread. To access the stack trace of the DatabaseConnectorThread, click on the icon in Figure 3.21 or the Stack Trace label adjacent to the icon. Figure 3.22 then appears.

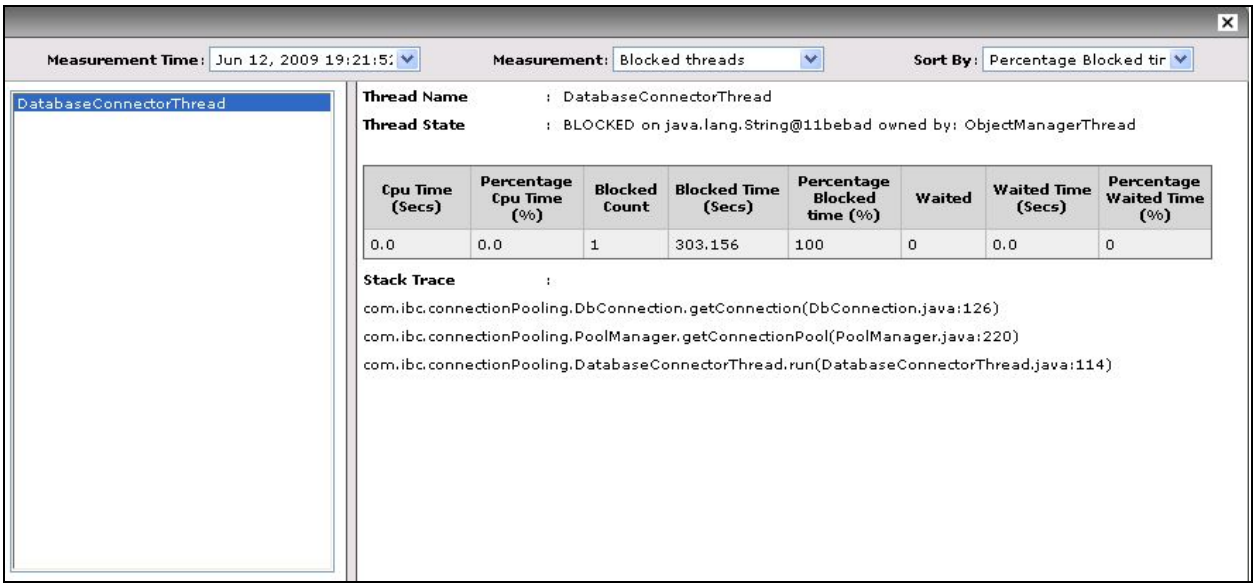


Figure 3.22: The Stack Trace of the blocked thread

While the left panel of Figure 3.22 displays the DatabaseConnectorThread, the right panel provides the following information about the DatabaseConnectorThread:

- The Thread State indicating the thread that is blocking the DatabaseConnectorThread, and the object on which the block occurred; from the right panel of Figure 3.22, we can infer that the DatabaseConnectorThread has been blocked on the java.lang.Strin@11bebad object owned by the ObjectManagerThread.
- The CPU usage of the DatabaseConnectorThread, and the number of times and duration for which this thread has been blocked and has been in waiting;
- The Stack Trace of the DatabaseConnectorThread.

Now that we have identified the blocked thread, let us proceed to determine the root-cause for this block. For this purpose, the **Stack Trace** of the *DatabaseConnectorThread* needs to be analyzed. As stated earlier, the stack trace needs to be analyzed in the top-down manner to identify the method that could have caused the block. Accordingly, we can conclude that the first method in the **Stack Trace** in Figure 3.22 is most likely to have introduced the block. This method, as can be seen from Figure 21, executes the lines of code starting from line **126** contained within the Java program file named **DbConnection.java**. In all probability, the problem should exist in this code block only. Reviewing this code block can therefore shed more light on the reasons for the *DatabaseConnectorThread* getting blocked. Hence, let us first open the **DbConnection.java** file in an editor (see Figure 3.23).

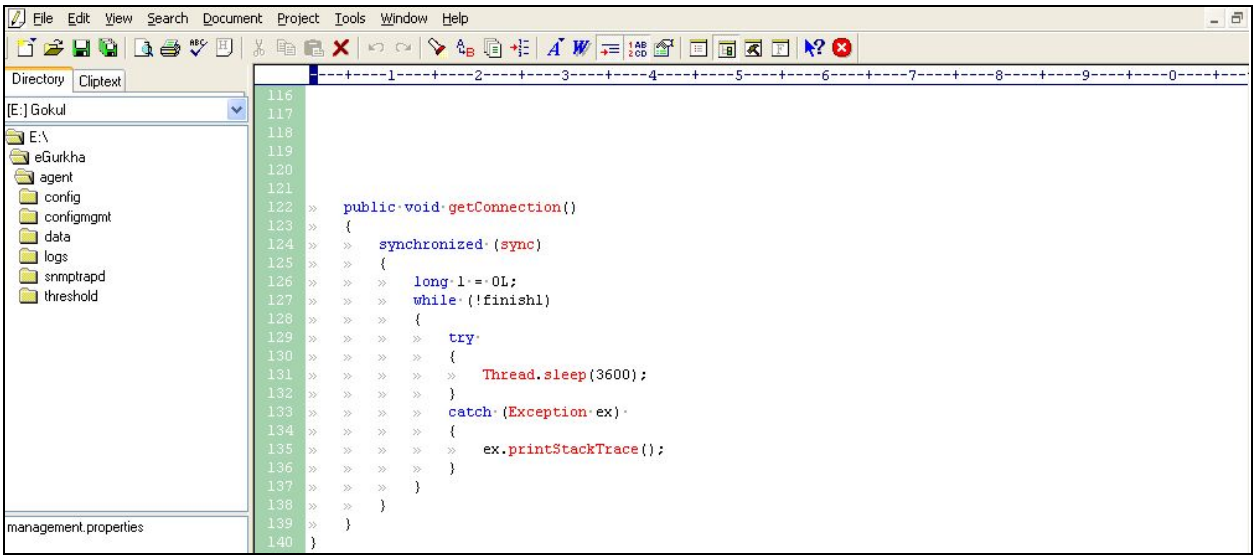


Figure 3.23: The DbConnection.java program file

Line 126 of Figure 3.23 is within a synchronized block. The object used to synchronize the accesses to this block is a variable named “sync”. Looking at the variable declarations at the top of the source code, we can see that the “sync” variable refers to the static string “test” (see Figure 37).

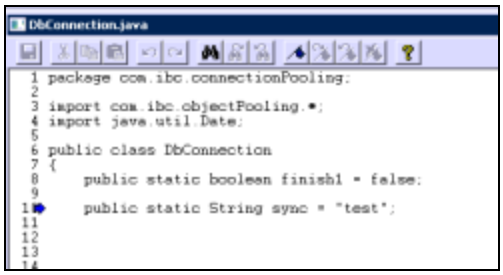


Figure 3.24: The lines of code preceding line 126 of the DbConnection.java program file

By comparing information form stack trace and the source we can see that the DatabaseConnectorThread is stuck entering the synchronized block. Access to the synchronized block is exclusive – so some other thead is blocking this DatabaseConnectorThread from entering the synchronized block. Looking at the stack trace again (see Figure 35), we can see the name of the blocking thread. The blocking thread is the thread named “ObjectManagerThread”.

We can now use the stack trace tool again to see the stack trace of the blocking ObjectManagerThread.

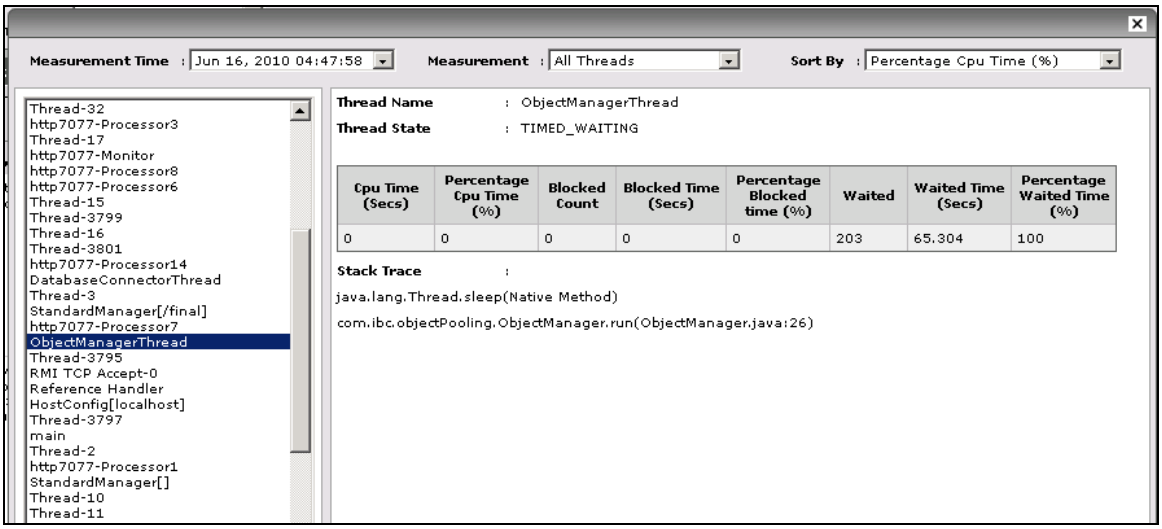


Figure 3.25: Viewing the stack trace of the ObjectManagerThread

From here, we can see that the ObjectManagerThread went into a timed waiting state at line number 26 of the ObjectManager.java source code.

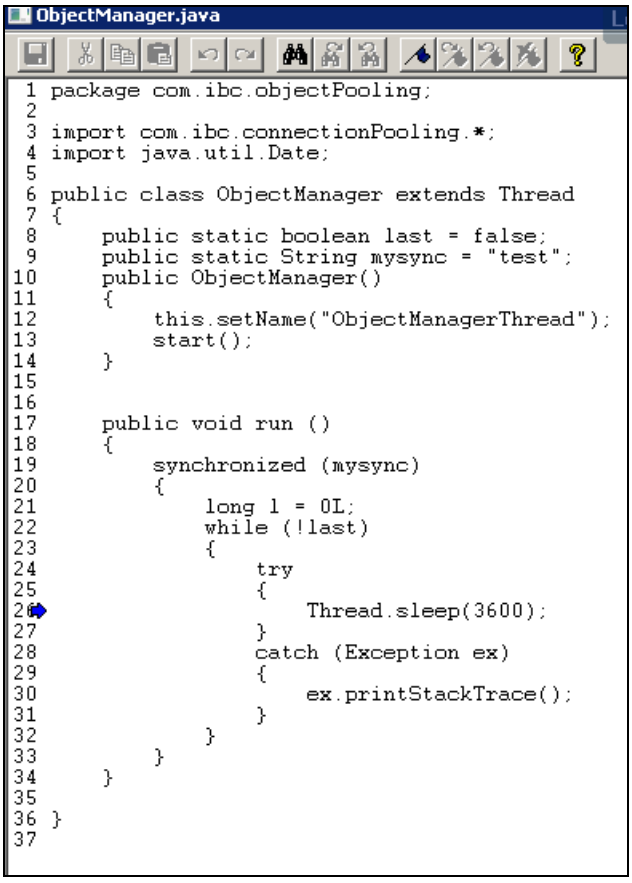


Figure 3.26: The lines of code in the ObjectManager.java source file

Again, using a text editor, we can see that the ObjectManager thread enters a 3600 second timed wait at line 26. This sleep call is inside a synchronized block with the local variable “mysync” being used as the object to synchronize on.

The key to troubleshooting this problem is to look at the variable declarations at the top of each source code file.

On the surface, it is not clear why the ObjectManager thread, which synchronizes a block using a variable called “mysync” which is local to this class would be blocked by the DbConnection thread, which synchronizes on a variable called “sync” that is local to the DbConnection class.

An astute java programmer, however, would know to look at the variable declarations at the top of each source code file. In that way, one will quickly observe that both the "mysync" variable of the ObjectManager class and the "sync" variable of the DbConnection class in fact refer to the same static string: “test”.

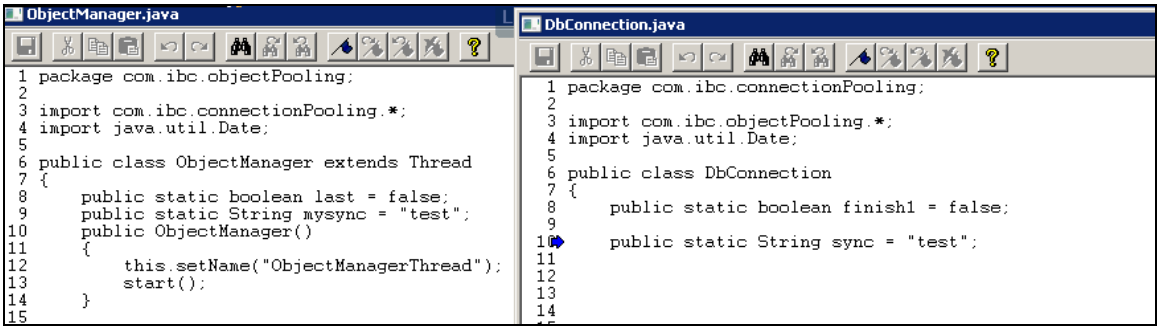


Figure 3.27: Comparing the ObjectManager and DbConnection classes

So, even though the programmer has given two different variable names in the two classes, the two classes refer to and are synchronizing on the same static string object “test”. This is why two unrelated threads are interfering with each other’s execution.

Modifying the two classes – ObjectManager and DbConnection – so that the variables "mysync" and "sync" point to two different strings by using the new object creator resolves the problem in this case.

We have demonstrated here a real-world example, where because of the careless use of variables, one could end up in a scenario where one thread blocks another. The solution in this case to avoid this problem is to define non-static variables that the two classes can use for synchronization. This example has demonstrated how the eG Java Monitor can help diagnose and resolve a complex multi-thread synchronization problem in a Java application.

3.4.3 Identifying and Diagnosing a Thread Waiting Situation in the JVM

This section takes the help of the sapbusiness application yet again to demonstrate how the eG JVM Monitor quickly isolates waiting threads and identifies the root-cause for the thread waits.

Whenever a thread goes into waiting, the value of the Waiting threads measure reported by the jvm Threads test mapped to the jvm Internals layer gets incremented by 1 (see Figure 3.28).

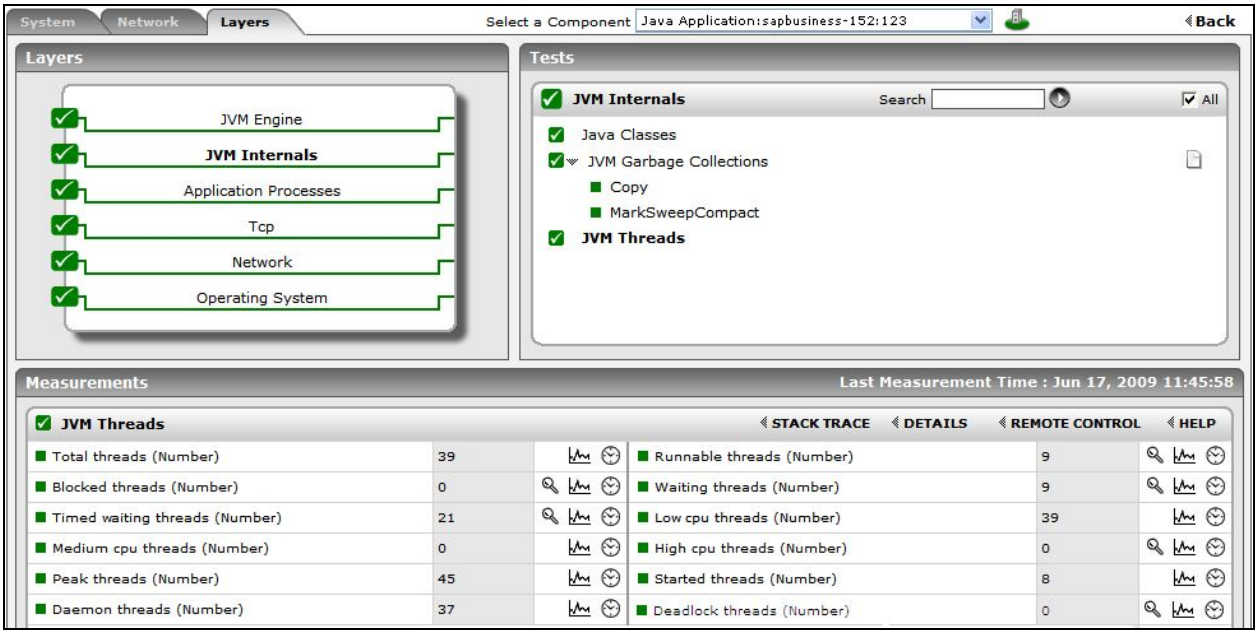
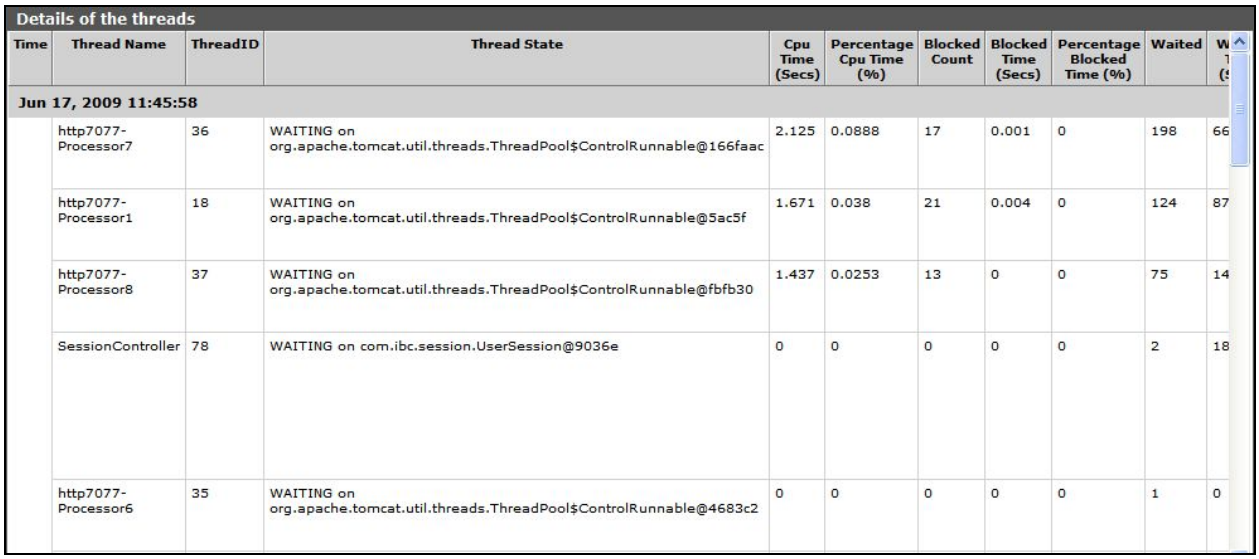


Figure 3.28: The Waiting threads

To know which threads are in waiting, click on the **DIAGNOSIS** icon corresponding to the Waiting threads measure in Figure 3.28. Figure 3.29 then appears listing all the threads that are currently in waiting.

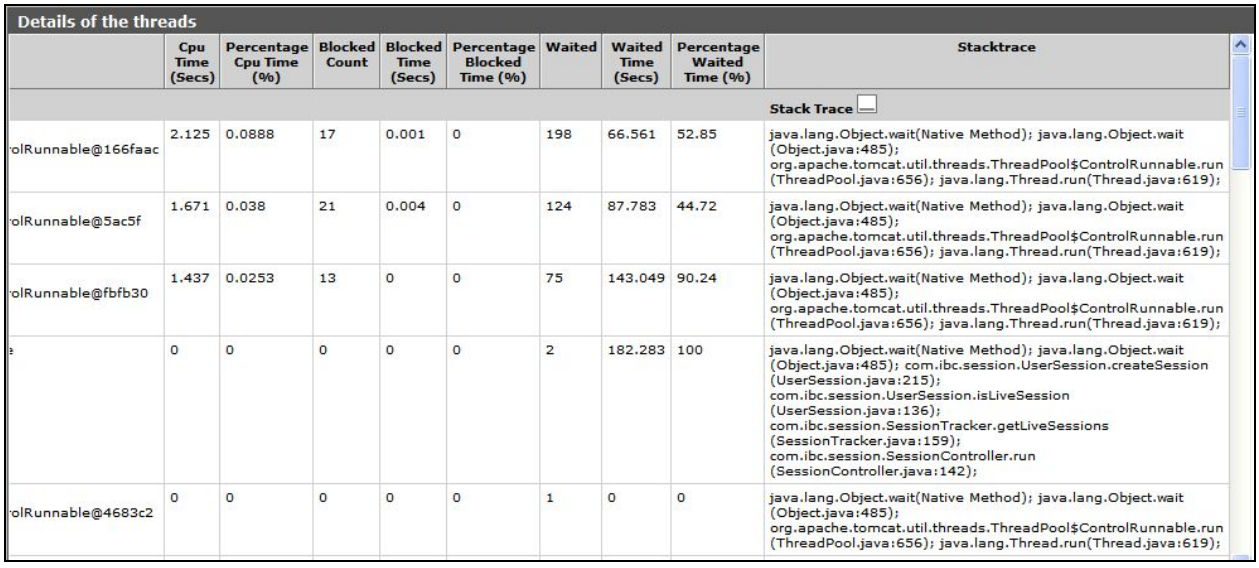


Details of the threads

Time	Thread Name	ThreadID	Thread State	Cpu Time (Secs)	Percentage Cpu Time (%)	Blocked Count	Blocked Time (Secs)	Percentage Blocked Time (%)	Waited	W
Jun 17, 2009 11:45:58										
	http7077-Processor7	36	WAITING on org.apache.tomcat.util.threads.ThreadPool$ControlRunnable@166faac	2.125	0.0888	17	0.001	0	198	66
	http7077-Processor1	18	WAITING on org.apache.tomcat.util.threads.ThreadPool$ControlRunnable@5ac5f	1.671	0.038	21	0.004	0	124	87
	http7077-Processor8	37	WAITING on org.apache.tomcat.util.threads.ThreadPool$ControlRunnable@fbfb30	1.437	0.0253	13	0	0	75	14
	SessionController	78	WAITING on com.ibc.session.UserSession@9036e	0	0	0	0	0	2	18
	http7077-Processor6	35	WAITING on org.apache.tomcat.util.threads.ThreadPool$ControlRunnable@4683c2	0	0	0	0	0	1	0

Figure 3.29: The detailed diagnosis of the Waiting threads measure

Of the threads listed in Figure 3.29, those that begin with http* are Tomcat's java threads. For these threads to be in a waiting state is normal, and hence, these threads can be ignored. Only the SessionController thread indicated by Figure 3.29 is an application-specific thread. To know why this thread has been in waiting, you need to study the stack trace of the thread; for this, first scroll to the left of Figure 3.29. You will then be able to view the stack trace of the thread.



Details of the threads

	Cpu Time (Secs)	Percentage Cpu Time (%)	Blocked Count	Blocked Time (Secs)	Percentage Blocked Time (%)	Waited	Waited Time (Secs)	Percentage Waited Time (%)	Stacktrace
									Stack Trace
olRunnable@166faac	2.125	0.0888	17	0.001	0	198	66.561	52.85	java.lang.Object.wait(Native Method); java.lang.Object.wait (Object.java:485); org.apache.tomcat.util.threads.ThreadPool$ControlRunnable.run (ThreadPool.java:656); java.lang.Thread.run(Thread.java:619);
olRunnable@5ac5f	1.671	0.038	21	0.004	0	124	87.783	44.72	java.lang.Object.wait(Native Method); java.lang.Object.wait (Object.java:485); org.apache.tomcat.util.threads.ThreadPool$ControlRunnable.run (ThreadPool.java:656); java.lang.Thread.run(Thread.java:619);
olRunnable@fbfb30	1.437	0.0253	13	0	0	75	143.049	90.24	java.lang.Object.wait(Native Method); java.lang.Object.wait (Object.java:485); org.apache.tomcat.util.threads.ThreadPool$ControlRunnable.run (ThreadPool.java:656); java.lang.Thread.run(Thread.java:619);
	0	0	0	0	0	2	182.283	100	java.lang.Object.wait(Native Method); java.lang.Object.wait (Object.java:485); com.ibc.session.UserSession.createSession (UserSession.java:215); com.ibc.session.UserSession.isLiveSession (UserSession.java:136); com.ibc.session.SessionTracker.getLiveSessions (SessionTracker.java:159); com.ibc.session.SessionController.run (SessionController.java:142);
olRunnable@4683c2	0	0	0	0	0	1	0	0	java.lang.Object.wait(Native Method); java.lang.Object.wait (Object.java:485); org.apache.tomcat.util.threads.ThreadPool$ControlRunnable.run (ThreadPool.java:656); java.lang.Thread.run(Thread.java:619);

Figure 3.30: Viewing the stack trace of the waiting thread

If you want to view the stack trace more clearly, click on the icon in Figure 3.30 or the **Stack Trace** label adjacent to the icon. Figure 3.31 then appears.

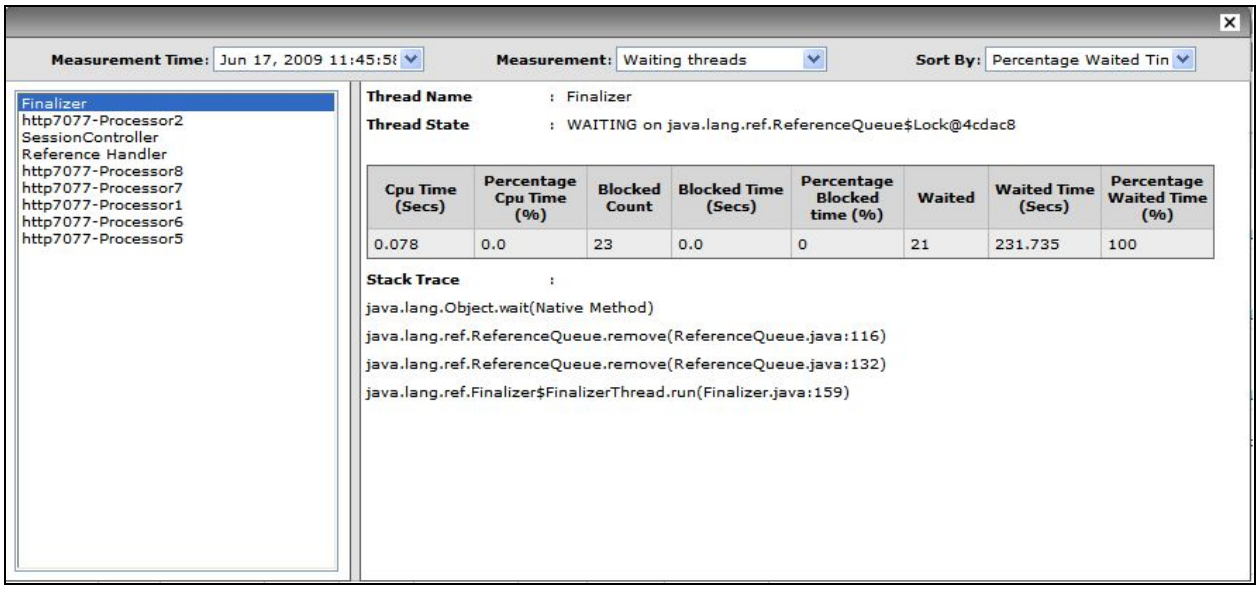


Figure 3.31: The Thread Diagnosis window for Waiting threads

The left panel of Figure 3.31 lists all the waiting threads, with the thread that registered the highest waiting time being selected by default. Since we are interested in the user-defined **SessionController** thread, select it from the left panel. The right panel will then change as depicted by Figure 3.32 below.

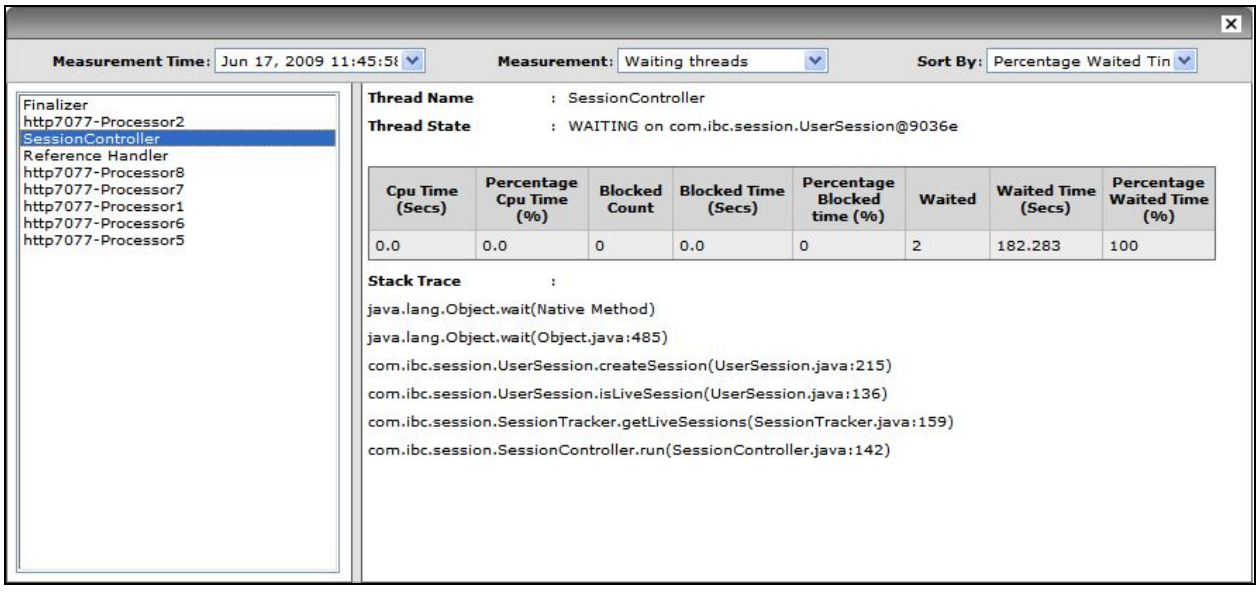


Figure 3.32: The stack trace for the SessionController thread

A close look at the stack trace reveals that the thread could have gone into the waiting mode while executing the code block starting at line **215** of the **UserSession.java** program file. To zero-in on the

precise code that could have caused the thread to wait, open the **UserSession.java** file in an editor, and locate line **215** in it.

```
210    public synchronized void createSession()
211    {
212        try
213        {
214            //System.out.println("Started to wait ... @ "+new java.util.Date());
215            wait();
216        }
217        catch (InterruptedException e)
218        {
219            System.out.println("Exiting MainThread...");
220        }
221    }
222
223    public synchronized void notifyThread()
224    {
225        try
226        {
227            notify();
228        }
229        catch (Exception e)
230        {
231            e.printStackTrace();
232        }
233    }
234 }
235
```

Figure 3.33: The UserSession.java file

The code block starting at line 215 of Figure 3.33 explicitly puts the thread in the wait state until such time that the notify() method is called to change the wait state to a runnable state. This piece of code will have to be optimized to reduce or even completely eliminate the waiting period of the **SessionController** thread.

With that, we have demonstrated the eG JVM Monitor's ability to detect waiting threads and lead you to the precise line of code that could have put the threads in a wait state.

3.4.4 Identifying and Diagnosing a Thread Deadlock Situation in the JVM

In this section, the sapclient application is used one more time to explain how the eG JVM Monitor can be used to report on deadlock situations in your JVM, and to diagnose the root-cause of the deadlock.

Until a deadlock situation arises, the **Deadlock threads** measure reported by the **JVM Threads** test will report only *0* as its value (see Figure 3.34).

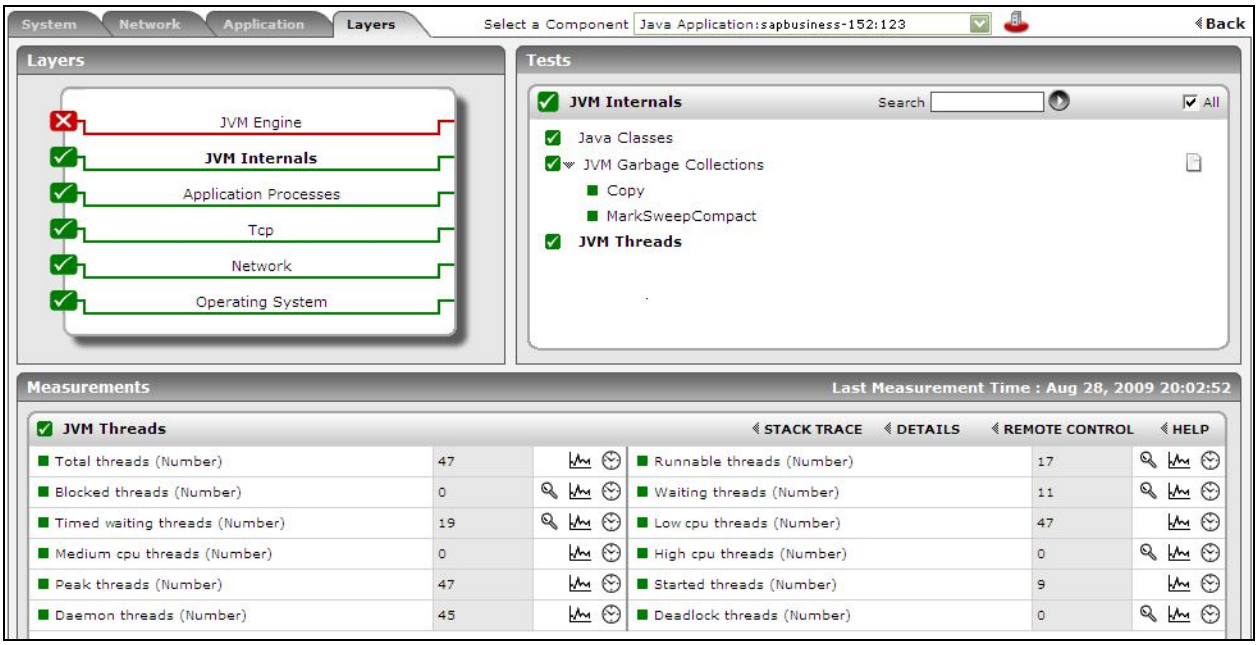


Figure 3.34: The JVM Threads test reporting 0 Deadlock threads

When, say 2 threads are deadlocked for a particular resource/object, then the **Deadlock threads** measure will report the value *2*, as depicted by Figure 3.35. Since a deadlock situation arises when two/more threads try to **block** each other from accessing a memory object or a resource, the value of the **Blocked threads** measure too will increase in the event of a deadlock; in the case of our example therefore, you will find that the **Blocked threads** measure too reports the value *2*.

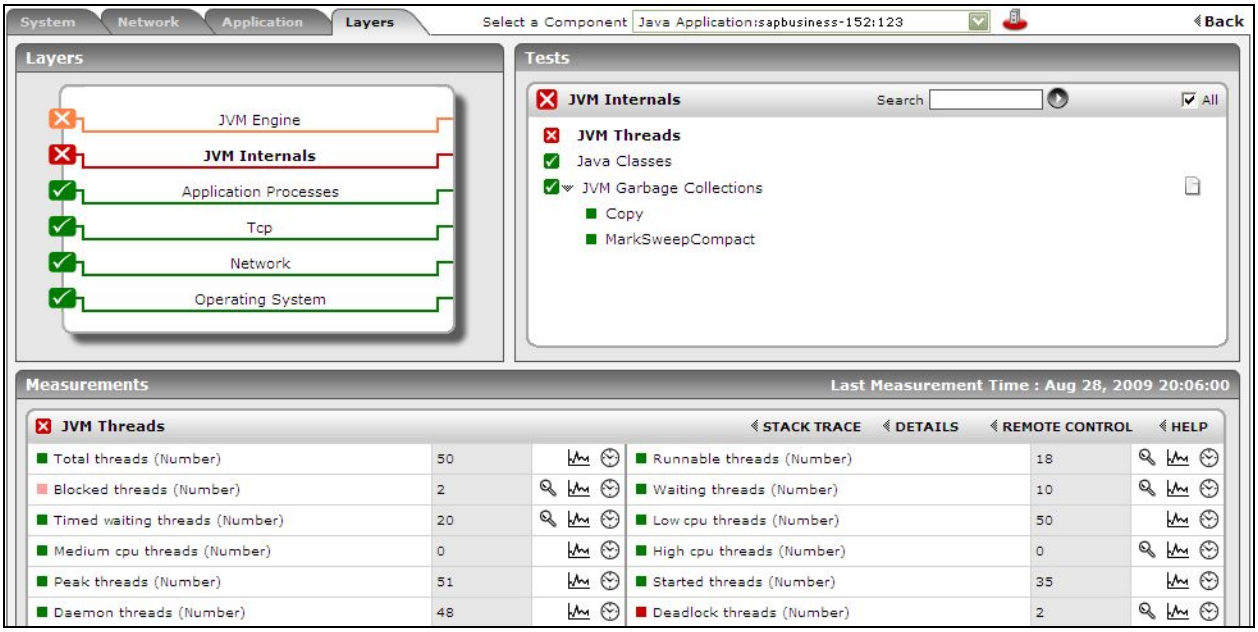
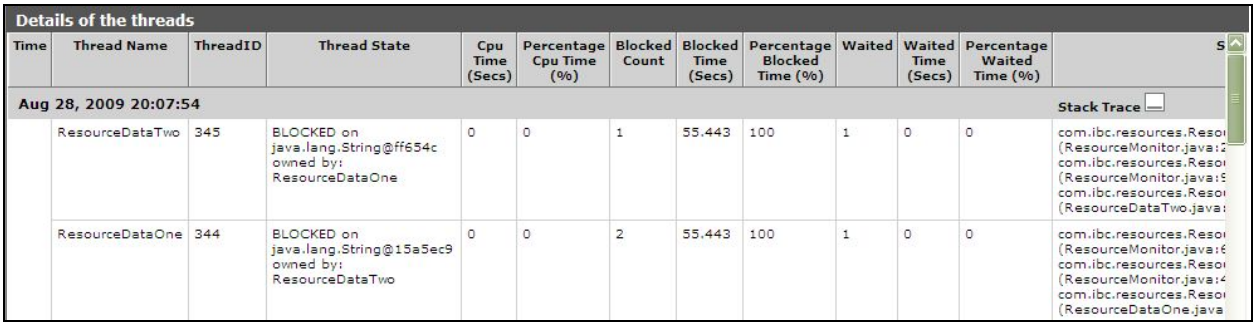


Figure 3.35: The Deadlock threads measure value increasing in the event of a deadlock situation

To know which threads are in a deadlock, click on the **DIAGNOSIS** icon corresponding to the **Deadlock threads** measure. Figure 3.36 then appears.

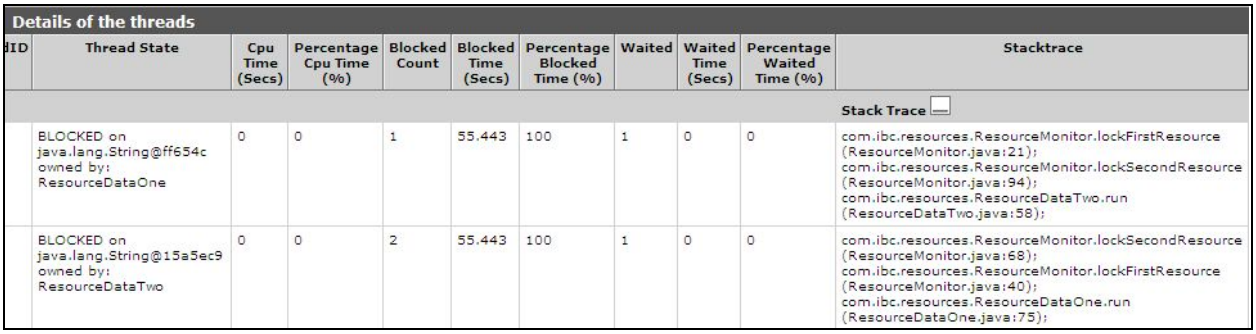


Details of the threads

Time	Thread Name	ThreadID	Thread State	Cpu Time (Secs)	Percentage Cpu Time (%)	Blocked Count	Blocked Time (Secs)	Percentage Blocked Time (%)	Waited	Waited Time (Secs)	Percentage Waited Time (%)	S
Aug 28, 2009 20:07:54												Stack Trace
	ResourceDataTwo	345	BLOCKED on java.lang.String@ff654c owned by: ResourceDataOne	0	0	1	55.443	100	1	0	0	com.ibc.resources.Resou (ResourceMonitor.java:2 com.ibc.resources.Resou (ResourceMonitor.java:9 com.ibc.resources.Resou (ResourceDataTwo.java:
	ResourceDataOne	344	BLOCKED on java.lang.String@15a5ec9 owned by: ResourceDataTwo	0	0	2	55.443	100	1	0	0	com.ibc.resources.Resou (ResourceMonitor.java:6 com.ibc.resources.Resou (ResourceMonitor.java:4 com.ibc.resources.Resou (ResourceDataOne.java

Figure 3.36: The detailed diagnosis page revealing the deadlocked threads

Figure 3.36 clearly reveals that 2 threads, namely – the ResourceDataTwo and the ResourceDataOne thread- are in a deadlock currently. To figure out why these two threads are deadlocked, you would have to carefully review the stack trace of both these threads. For this purpose, scroll to the left of Figure 3.36 to view the stack trace clearly.



Details of the threads

JID	Thread State	Cpu Time (Secs)	Percentage Cpu Time (%)	Blocked Count	Blocked Time (Secs)	Percentage Blocked Time (%)	Waited	Waited Time (Secs)	Percentage Waited Time (%)	Stacktrace
										Stack Trace
	BLOCKED on java.lang.String@ff654c owned by: ResourceDataOne	0	0	1	55.443	100	1	0	0	com.ibc.resources.ResourceMonitor.lockFirstResource (ResourceMonitor.java:21); com.ibc.resources.ResourceMonitor.lockSecondResource (ResourceMonitor.java:94); com.ibc.resources.ResourceDataTwo.run (ResourceDataTwo.java:58);
	BLOCKED on java.lang.String@15a5ec9 owned by: ResourceDataTwo	0	0	2	55.443	100	1	0	0	com.ibc.resources.ResourceMonitor.lockSecondResource (ResourceMonitor.java:68); com.ibc.resources.ResourceMonitor.lockFirstResource (ResourceMonitor.java:40); com.ibc.resources.ResourceDataOne.run (ResourceDataOne.java:75);

Figure 3.37: Viewing the stack trace of the dadlocked threads in the detailed diagnosis page

To keenly focus on the stack trace, without being distracted by the other columns in Figure 3.36 and Figure 3.37, click on the icon in Figure 3.37 or the **Stack Trace** label adjacent to the icon. Figure 3.38 then appears.

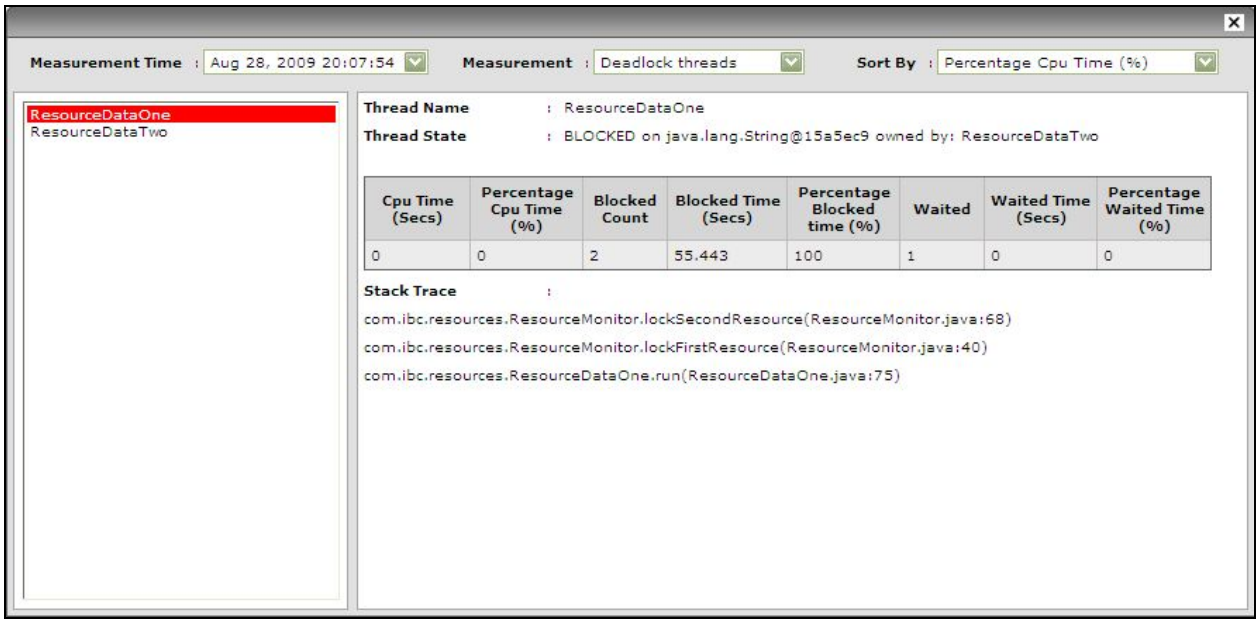


Figure 3.38: The stack trace for the ResourceDataOne thread

The left panel of Figure 3.38 lists the 2 deadlocked threads, with the thread that is the leading CPU consumer being selected by default – in the case of our example, this is the ResourceDataOne thread. For this default selection, the contents of the right panel will be as depicted by Figure 3.38 above. From the **Thread** State, it is evident that the ResourceDataOne thread has been blocked on an object that is owned by the ResourceDataTwo thread.

If you closely scrutinize the stack trace of ResourceDataOne, you will uncover that once the thread started running, it executed line 40 of the ResourceMonitor.java program file, which in turn invoked line 68 of the same file; the deadlock appears to have occurred at line 68 only.

Let us now shift our focus to the ResourceDataTwo thread. To view the stack trace of this thread, click on the thread name in the left panel of Figure 3.38. As you can see, the Thread State clearly indicates that the ResouceDataTwo thread has been blocked by the ResourceDataOne thread. With that, we can conclude that both threads are blocking each other, thus making for an ideal deadlock situation.

Analysis of the stack trace of the ResourceDataTwo thread (see Figure 3.39) reveals that once started, the thread executed line 94 of the ResourceMonitor.java file, which in turn invoked line 21 of the same file; since no lines of code have been executed subsequently, we can conclude that the deadlock occurred at line 21 only.

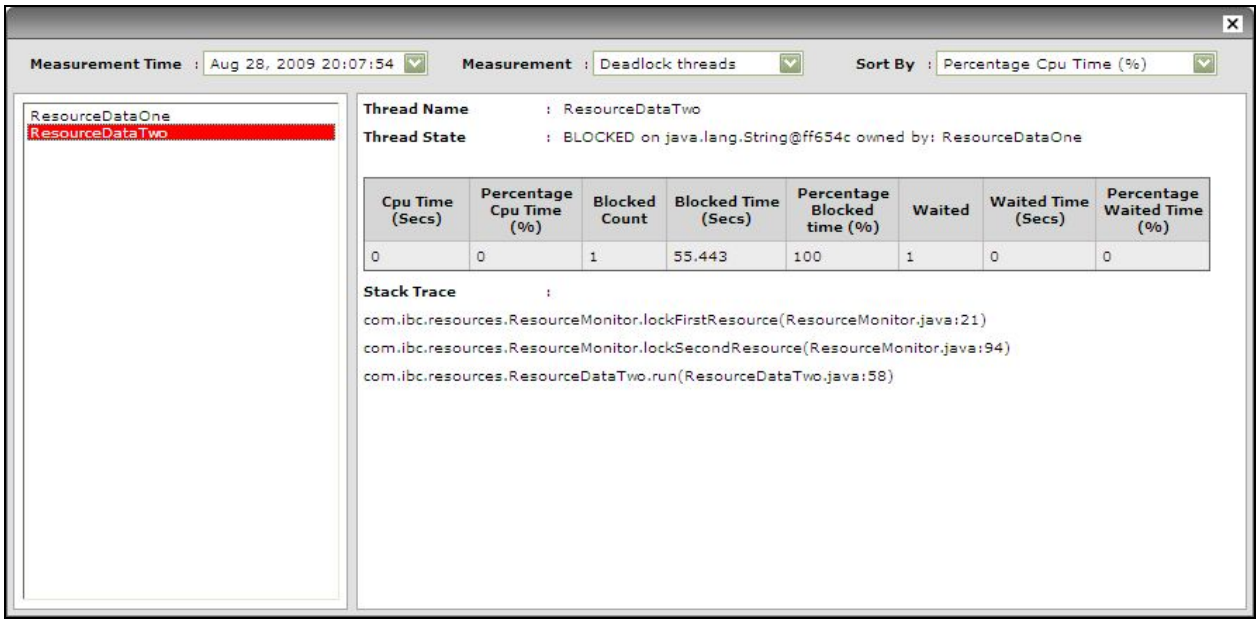


Figure 3.39: The stack trace for the ResourceDataTwo thread

From the above discussion, we can infer both the threads deadlocked while attempting to execute code contained within the ResourceMonitor.java file. We now need to examine the code in this file to figure out why the deadlock occurred. Let us therefore open the ResourceMonitor.java file.

```
61
62      public void lockSecondResource()
63      {
64          synchronized (resource2)
65          {
66              try
67              {
68                  Thread.sleep(500);
69              }
70              catch (InterruptedException e)
71              {
72                  e.printStackTrace();
73              }
74              lockFirstResource();
75          }
76      }
77  }
```

Figure 3.40: The lines of code executed by the ResourceDataOne thread

If you can recall, the stack trace of the ResourceDataOne thread indicated a problem while executing the code around line number 68 (see Figure 3.38) of the ResourceMonitor.java file. Figure 3.40 depicts this piece of code. According to this code, the ResourceDataOne thread calls a lockSecondResource() method, which in turn invokes a synchronized block that puts the thread to

sleep for 500 milliseconds; a synchronized method, when called by a thread, cannot be invoked by any other thread until its original caller releases the method.

Going back to Figure 3.40, at the end of the sleep duration of 500 milliseconds, the *synchronized* block will invoke another method named *lockFirstResource ()*. However, note that this method and the *lockSecondResource()* method are also called by the *ResourceDataTwo* thread. To verify this, let us proceed to review the lines of code executed by the *ResourceDataTwo* thread (see Figure 3.41).

```
13
14
15      public void lockFirstResource()
16      {
17          synchronized (resource1)
18          {
19              try
20              {
21                  Thread.sleep(500);
22              }
23              catch (InterruptedException e)
24              {
25                  e.printStackTrace();
26              }
27              lockSecondResource();
28          }
29      }
```

Figure 3.41: The lines of code executed by the ResourceDataTwo thread

As per the stack trace corresponding to the ResourceDataTwo thread (see Figure 3.39), the deadlock creeps in at line 21 of the ResourceMonitor.java file. Figure 3.41 depicts the code around line 21 of the ResourceMonitor.java file. This code reveals that the ResourceDataTwo thread executes a lockFirstResource()method, which in turn invokes a synchronized block; within this block, the thread is put to sleep for 500 milliseconds. Once the sleep ends, the block will invoke the lockSecondResource() method; both this method and the lockFirstResource() method are also executed by the ResourceDataOne thread.

From the discussion above, the following are evident:

- The ResourceDataOne thread will not be able to execute the lockSecondResource() method, since the ResourceDataTwo thread calls this method within a synchronized block – this implies that the ResourceDataTwo thread will 'block' the ResourceDataOne thread from executing the lockSecondResource() method until such time that ResourceDataTwo executes the method.
- The ResourceDataTwo thread on the other hand, will not be able to execute the lockFirstResource() method, since the ResourceDataOne thread calls this method within a synchronized block – this implies that the ResourceDataOne thread will 'block' the ResourceDataTwo thread from executing the lockFirstResource() method until such time that ResourceDataOne executes the method.

Since both threads keep blocking each other, a deadlock situation occurs.

With that, we have demonstrated the eG JVM Monitor's ability to detect deadlock threads and lead you to the precise line of code that could have caused the deadlock.

3.4.5 Identifying and Diagnosing Memory Issues in the JVM

This section takes the example of the *sapclient* application again to demonstrate the effectiveness of the eG JVM Monitor in proactively detecting and alerting administrators to memory contentions experienced by Java applications.

If the usage of a memory pool increases, the eG JVM Monitor indicates the same using the Used memory measure for that pool reported by the **JVM Memory Usage** test mapped to the **JVM Engine** layer.

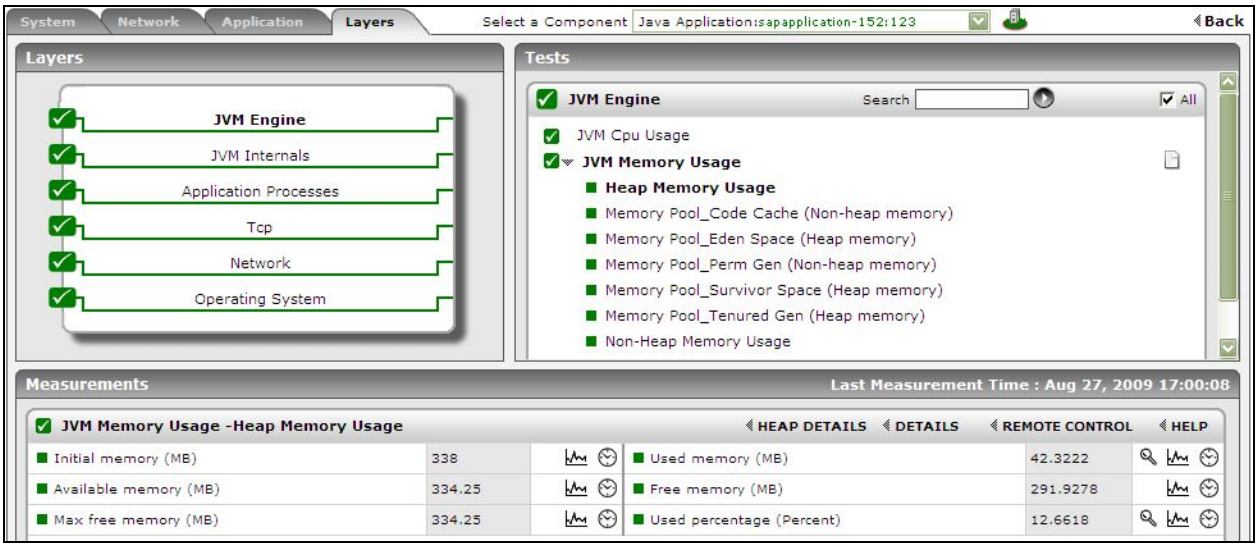


Figure 3.42: The Used memory measure indicating the amount of pool memory being utilized

To know which class is consuming memory excessively, click on the **DIAGNOSIS** icon corresponding to the Used memory measure in Figure 3.42. Figure 3.43 then appears listing all the classes that are using the pool memory, the amount and percentage of memory used by each class, the number of instances of each class that is currently operational, and also the percentage of currently running instances of each class. Since this list is by default sorted in the descending order of the percentage memory usage, the first class in the list will obviously be the leading memory consumer. In the case of our example, the memory contention in the sapbusiness application has been caused by the 22% heap memory usage of the com.ibc.object.SapBusinessObject class.

Details of JVM Heap Usage

Time	Class Name	Instance Count	Instance Percentage	Memory used(MB)	Percentage memory used
Aug 27, 2009 17:00:08					Heap Details
	com.ibc.object.SapBusinessObject	129729	14.1368	13.5668	21.3516
	[Ljava.lang.Object;	35391	3.8566	9.8693	15.5324
	<constMethodKlas>	42178	4.5962	5.9795	9.4105
	[B	7884	0.8591	5.9332	9.3378
	java.lang.String	209735	22.8552	4.8005	7.555
	<methodKlas>	42178	4.5962	3.2229	5.0722
	[I	9541	1.0397	2.9486	4.6406
	<symbolKlas>	70245	7.6547	2.8693	4.5158
	[C	180000	19.6149	2.7466	4.3226
	<constantPoolKlas>	3167	0.3451	2.0846	3.2808
	<instanceKlassKlas>	3167	0.3451	1.3359	2.1025
	<constantPoolCacheKlas>	2745	0.2991	1.2858	2.0235
	java.util.Hashtable$Entry	27440	2.9902	0.6281	0.9884
	java.util.Vector	26135	2.848	0.5982	0.9414
	<methodDataKlas>	1163	0.1267	0.5373	0.8456
	[Ljava.util.Hashtable$Entry;	5713	0.6226	0.4688	0.7378
	[S	4552	0.496	0.352	0.5539
	[Ljava.util.HashMap$Entry;	4453	0.4853	0.3435	0.5405

Figure 3.43: The detailed diagnosis of the Used memory measure

Sometimes, you might want to sort the classes by another column or quickly switch to another measurement period to analyze the memory usage during that time frame. To achieve this, click on the **Heap Details** link or the button next to it. Figure 53 then appears, allowing you the flexibility to view memory-consuming classes based on a **Sort by** option and a **Measurement Time** of your choice.

Measurement Time : Aug 27, 2009 17:00:08 Sort By : Percentage memory used

Class Name	Instance Count	Instance Percentage	Memory used	Percentage memory used
com.ibc.object.SapBusinessObject	129729	14.1368	13.5668	21.3516
[Ljava.lang.Object;	35391	3.8566	9.8693	15.5324
<constMethodKlas>	42178	4.5962	5.9795	9.4105
[B	7884	0.8591	5.9332	9.3378
java.lang.String	209735	22.8552	4.8005	7.555
<methodKlas>	42178	4.5962	3.2229	5.0722
[I	9541	1.0397	2.9486	4.6406
<symbolKlas>	70245	7.6547	2.8693	4.5158
[C	180000	19.6149	2.7466	4.3226
<constantPoolKlas>	3167	0.3451	2.0846	3.2808
<instanceKlassKlas>	3167	0.3451	1.3359	2.1025
<constantPoolCacheKlas>	2745	0.2991	1.2858	2.0235
java.util.Hashtable$Entry	27440	2.9902	0.6281	0.9884
java.util.Vector	26135	2.848	0.5982	0.9414
<methodDataKlas>	1163	0.1267	0.5373	0.8456
[Ljava.util.Hashtable$Entry;	5713	0.6226	0.4688	0.7378
[S	4552	0.496	0.352	0.5539
[Ljava.util.HashMap$Entry;	4453	0.4853	0.3435	0.5405

Figure 3.44: Choosing a different Sory By option and Measurement Time

Careful examination of the method that calls the SapBusinessObject (see Figure 3.45) reveals that an endless while loop is causing an array list named a to be populated with 20,000 instances of the SapBusinessObject, every 10 seconds! The continuous addition of objects is quiet obviously depleting the memory available to the JVM.

```
115
116     public void getClonedObject()
117     {
118         while (!finish2)
119         {
120             ArrayList a = new ArrayList();
121             for(int i=0;i<20000;i++)
122             {
123                 SapBusinessObject sbo = new SapBusinessObject("java",i);
124                 a.add(sbo);
125             }
126             try
127             {
128                 Thread.currentThread().sleep(10000);
129             }
130             catch (Exception ex)
131             {
132                 ex.printStackTrace();
133             }
134         }
135     }
136
137 }
```

Figure 3.45: The method that is invoking the SapBusinessObject

This is how the eG JVM Monitor greatly simplifies the process of identifying the source of memory bottlenecks in a Java application.

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.