Monitoring JRun Application Server

eG Innovations Product Documentation

www.eginnovations.com

Total Performance Visibility

Table of Contents

Table of Figures

Chapter 1: Introduction

JRun is an application server from Macromedia that is based on Sun Microsystems' Java 2 Platform, Enterprise Edition (J2EE). JRun consists of Java Server Page (JSP), Java servlets, Enterprise JavaBeans, the Java Transaction Service (JTS), and the Java Messaging Service (JMS). JRun works with the most popular Web servers including Apache, Microsoft's Internet Information Server (IIS), and any other Web server that supports Internet Server Application Program Interface (ISAPI) or the Web's common gateway interface (CGI).

If the JRun application server is unable to process the business logic swiftly and efficiently, end-users are bound to experience a significant slowdown in the responsiveness of the web application they interact with. To ensure that the applications supported by the JRun server function optimally at all times, constant monitoring of the JRun server's availability and internal operations is essential.

Chapter 2: How to Monitor JRun Application Server Using eG Enterprise?

eG Enterprise monitors the JRun application servers in both agentless and agent-based manners. Before attempting JRun monitoring, a set of pre-requisites should be fulfilled to configure the JRun application server to work with eG agent. The following sections describe how to configure a JRun application server to work with eG agent.

2.1 Configuring a JRun application server 3.0

In order to configure a JRun application server for monitoring by an eG agent, you need to enable the monitoring logs and configure the **global.properties** for customizing the output of the monitoring logs, by following the steps given below:

1. Open the **global.properties** file that resides in the <JRUN_INSTALL_DIR>/lib directory.
2. Look for the entry **logging.loglevel** and alter its value to:

   ```
   logging.loglevel=info,warning,error,metrics
   ```

   This ensures that all information, warnings, errors and metrics are logged by the JRun server. The eG agent uses the logged metrics for collecting statistics related to a JRun server.

3. The eG agent requires the metrics reported by each JRun server instance to be logged to a separate file. To do so, set the **logging.filelogwriter.filename** property as shown below:

   ```
   logging.filelogwriter.filename={jrun.rootdir}/logs/{jrun.server.name}-{log.level}.log
   ```

   The above process creates a log file named <SERVER_NAME>-metrics.log, to which the measures generated during monitoring are logged. The eG agent reads the data available in this log file and displays the same in the monitor interface.

4. Next, set the value of the properties **logging.filelogwriter.rotationsize** and **logging.filelogwriter.rotationfiles** in the **global.properties** file to 100000 and 5 respectively, as shown below:

   ```
   logging.filelogwriter.rotationsize=100000
   logging.filelogwriter.rotationfiles=5
   ```

   The rotation size specifies the maximum size of a log file. The smaller the value, the lower the monitoring overhead.

5. Ensure that the **monitor.format** property is set to **{monitor.combined-format}**. Then, reconfigure the value of **monitor.combined-format** as shown below, to define the pattern in which the metrics are to be logged.

```
monitor.combined-format=(jcp+web)  {{jcp.busyTh}+{web.busyTh}} {{jcp.delayTh}+
{web.delayTh}} {{jcp.totalTh}+{web.totalTh}} {totalMemory} {freeMemory} {sessions}
{sessionsInMem} {{jcp.handledRq}+{web.handledRq}} {{jcp.handledMs}+{web.handledMs}}
{{jcp.bytesIn}+{web.bytesIn}} {{jcp.bytesOut}+{web.bytesOut}} {{jcp.delayMs}+
{web.delayMs}} {{jcp.delayRq}+{web.delayRq}} {{jcp.droppedRq}+{web.droppedRq}}
```

6. Then, set **monitor.interval** to a value (in seconds), which could be 80-90% of the test frequency. This value has to be very carefully set as all the measures that the agent reports would be relative to the frequency at which the logging is taking place.

   For example, if the eG agent executes all the related JRun tests at a frequency of 300 seconds, set the value of **monitor.interval** to 270 (300*0.09) as shown below:

```
monitor.interval=270
```

7. Finally, restart the **JRun default server** service and check whether the metrics that have logged into the **default-metrics.log** file are in the following format:

```
05/29 19:05:39 metrics (JRun) (jcp+web)  0 0 11 1984 747 0 0 25 340 6950 6852 0 0
05/29 19:05:49 metrics (JRun) (jcp+web)  0 0 11 1984 606 0 0 14 171 3892 3836 0 0 0
05/29 19:05:59 metrics (JRun) (jcp+web)  0 0 11 1984 523 0 0 0 0 0 0 0 0 0
05/29 19:06:09 metrics (JRun) (jcp+web)  0 0 11 1984 440 0 0 0 0 0 0 0 0 0
05/29 19:06:19 metrics (JRun) (jcp+web)  0 0 11 1984 358 0 0 0 0 0 0 0 0 0
```

2.2 Configuring a JRun Application Server 4.0

A single JRun server 4.0 may contain multiple server instances. The JRun application server 4.0 maintains individual logs for each of its server instances.

To monitor a specific server, modify the file **jrun.xml** which lies in the <JRUN_INSTALL_ DIR>/servers/<SERVER_NAME>/SERVER-INF directory, where JRUN_INSTALL_DIR is the home directory for the JRun server and SERVER_NAME is the name of the server instance. For example, if you have a server instance named APPLICATION 1, then the SERVER_NAME will be APPLICATION 1.

Make the following changes in this file:

1. Enable metrics logging by setting the **metricsEnabled** attribute of **LoggerService** to true as shown below.

```
<attribute name="metricsEnabled">true</attribute>
```

2. Next, specify the file name to which the metrics must be logged. For that, add **{log.level}** to the **filename** attribute of the **FileLogEventHandler** service, as shown by the following example:

   ```
   <service class="jrunx.logger.FileLogEventHandler" name="FileLogEventHandler">
   <attribute name="filename">{jrun.rootdir}/logs/{jrun.server.name}-
   {log.level}.log</attribute>
   </service>
   ```

   The above process creates a log file named <SERVER_NAME>-metrics.log, to which the measures generated during monitoring are logged. The eG agent reads the data available in this log file and displays the same in the monitor interface.

3. Set the **rotationSize** and **rotationFiles** attribute of the **FileLogEventHandler** service to 100k and 3 respectively as shown below:

   ```
   <attribute name="rotationSize">100k</attribute>
   <attribute name="rotationFiles">3</attribute>
   ```

   The rotation size specifies the maximum size of a log file. The smaller the value, the lower the monitoring overhead.

4. To define the format in which metrics are to be logged, make the following changes in the **metricsFormat** attribute of the **LoggerService**.

   ```
   <attribute name="metricsFormat">{{jrpp.busyTh}+{web.busyTh}} {{jrpp.delayTh}+
   {web.delayTh}} {{jrpp.totalTh}+{web.totalTh}} {totalMemory} {freeMemory} {sessions}
   {sessionsInMem} {{jrpp.handledRq}+{web.handledRq}} {{jrpp.handledMs}+{web.handledMs}}
   {{jrpp.bytesIn}+{web.bytesIn}} {{jrpp.bytesOut}+{web.bytesOut}} {{jrpp.delayMs}+
   {web.delayMs}} {{jrpp.delayRq}+{web.delayRq}} {{jrpp.droppedRq}+
   {web.droppedRq}}</attribute>
   ```

5. Then, set the **metricsLogFrequency** attribute of **LoggerService** to a value (in seconds), which could be 80-90% of the test frequency. This value has to be very carefully set as all the measures that the agent reports would be relative to the frequency at which the logging is taking place.

   For example, if the eG agent executes all the related JRun tests at a frequency of 300 seconds, set the value of metricsLogFrequency to 270 (300*0.09) as shown below:

   ```
   <attribute name="metricsLogFrequency">270</attribute>
   ```

6. Finally, restart the specific JRun server for which you have enabled metrics logging and check whether the metrics that have logged into the <SERVER_NAME>-metrics.log file are in the following format:

```
06/14 12:06:58 metrics 0 0 1 32576 29198 1 1 0 0 0 0 0 0 0
06/14 12:11:28 metrics 0 0 2 32576 28310 1 1 18 17646 4535 2845 0 0 0
06/14 12:15:58 metrics 0 0 2 32576 29918 1 1 1 151 308 254 0 0 0
06/14 12:20:28 metrics 0 0 1 32576 29892 1 1 0 0 0 0 0 0 0
```

2.3 Configuring the eG Agent to Collect JVM-related Metrics from the JRun Server 3.0

The **JVM** layer of the *JRun* monitoring model is mapped to tests that report critical statistics related to the JRun server's JVM. These statistics typically reveal the following:

- The count of classes loaded/unloaded (Java Classes test)
- JVM thread usage (**JVM Threads** test)
- CPU and memory usage of the JVM (**JVM Cpu Usage** test and **JVM Memory Usage** test)
- The effectiveness of the JVM's garbage collection activity (**JVM Garbage Collections** test)
- The uptime of the JVM (JVM Uptime test)
- Whether JMX is currently enabled/disabled on the target WebLogic server (**JMX Connection to JVM** test)
- The count and status of file descriptors (**JVM File Descriptors** test)

These tests connect to the JRE used by the JRun application server to pull out the above-mentioned metrics. For these tests to execute, you should configure the eG agent to connect to JRE and collect the required metrics, using either of the following methodologies:

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

Since both JMX and SNMP support are available for JRE 1.5 and above only, these tests **will work only if the JRun server being monitored uses JRE 1.5 and above**.

If you choose to use JMX for pulling out the desired metrics from the JRE, then the following broad steps need to be followed before configuring these tests:

1. First, determine whether the JMX requires no authentication at all, or requires authentication (but no security)
2. If JMX does not require authentication, follow the steps below:

- Login to the target JRun server.
- Edit the *management.properties* file that is used by the JRE of the target JRun server, and configure the following in it:
- The JMX remote port
- Whether JMX is SSL-enabled or not
- Whether JMX requires authentication or not
- To know how to configure these, refer to the *Monitoring Java Applications* document.
- Save the file.

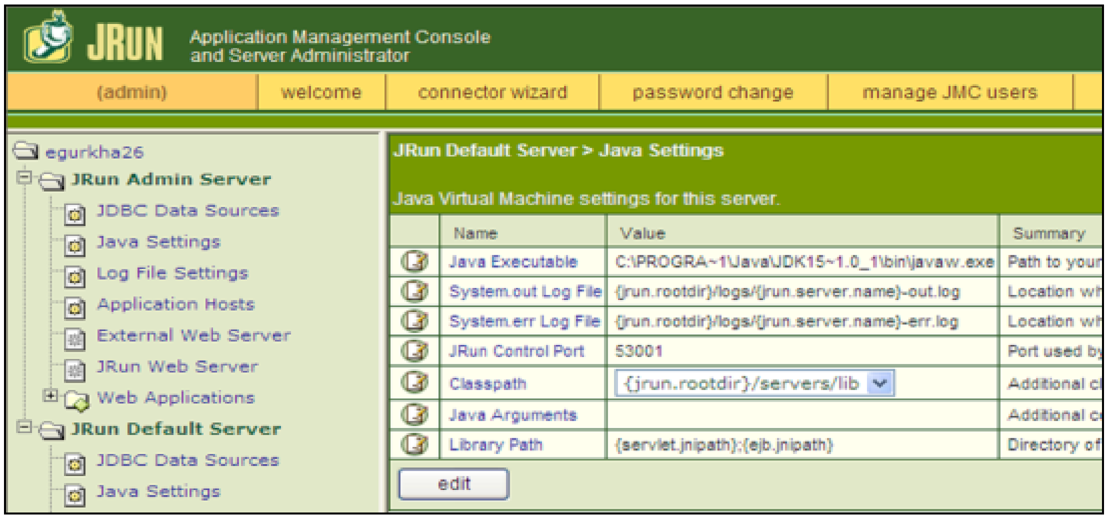
3. If the JMX requires authentication (but no security), follow the steps below:

   - Login to the target JRun server. If the server is executing on a Windows host, then, login to the host as a local/domain administrator.
   - Next, copy the *jmxremote.password.template* file in the <JAVA_HOME>\jre\lib\management folder 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.
   - 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 *Monitoring Java Applications* document.
   - Then, proceed to make the *jmxremote.password* file secure by granting a single user “full access” to that file. To know how to achieve this, refer to the *Monitoring Java Applications* document.
   - Edit the *management.properties* file that is used by the JRE of the target JRun server, and configure the following in it:

     - The JMX remote port
     - Whether JMX is SSL-enabled or not
     - Whether JMX requires authentication or not
     - The full path to the *jmxremote.access* file
     - The full path to the *jmxremote.password* file

     To know how to configure these, refer to the *Monitoring Java Applications* document.
   - Then, save the file.

- Next, connect to the JRun application management console using the URL: **http://<JRunServerIP>:<JRunPort>**
- The default port number of the JRun server is 8000.
- Provide the credentials for logging into the console, if prompted.
- When the console opens, you will find a tree-structure in the left panel of the console. Follow the node sequence, *JRun Default Server -> Java Settings*, in the tree-structure.



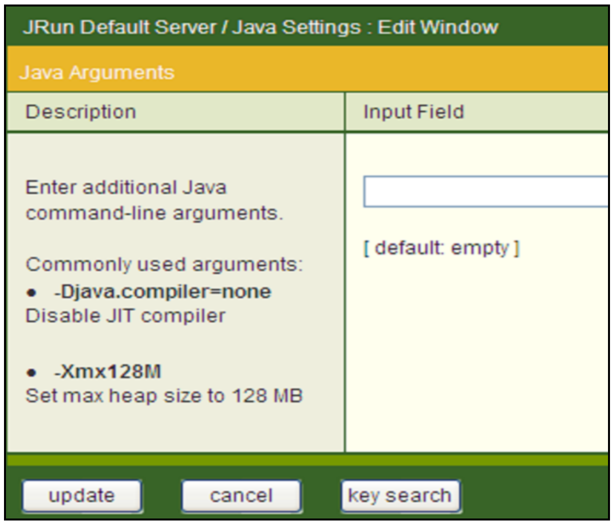
JRun Default Server > Java Settings

Java Virtual Machine settings for this server.

Name	Value	Summary
Java Executable	C:\PROGRA~1\Java\JDK15~1.0_1\bin\javaw.exe	Path to your
System.out Log File	{jrun.rootdir}/logs/{jrun.server.name}-out.log	Location wh
System.err Log File	{jrun.rootdir}/logs/{jrun.server.name}-err.log	Location wh
JRun Control Port	53001	Port used by
Classpath	{jrun.rootdir}/servers/lib	Additional cl
Java Arguments		Additional c
Library Path	{servlet.jnipath};{ejb.jnipath}	Directory of

Figure 2.1: Viewing the Java Virtual Machine Settings for the target JRun server

- The Java Virtual Machine settings will then appear in the right panel (see Figure 2.1). Click on the icon preceding the **Java Arguments** entry in the right panel.


JRun Default Server / Java Settings : Edit Window

Java Arguments

Description	Input Field
Enter additional Java command-line arguments. Commonly used arguments: • **-Djava.compiler=none** Disable JIT compiler • **-Xmx128M** Set max heap size to 128 MB	[default: empty]

Figure 2.2: Configuring the Java Arguments

- In the **Input Field** text box of Figure 2.2 that appears, enter the following:

```
-Dcom.sun.management.config.file=<Path to management.properties file used the JRE of
the target JRun server>
```

- Finally, stop and start the JRun server.

Note:

To know how to enable SNMP support for the JRE, refer to the *Monitoring Java Applications* document.

Once the above-said pre-requisistes are fulfilled, proceed to manage the JRun application server using the eG administrative interface. The steps for managing the server have been discussed in the below section.

Chapter 2: How to Monitor JRun Application Server Using eG Enterprise?

eG Enterprise monitors the JRun application servers in both agentless and agent-based manners. Before attempting JRun monitoring, a set of pre-requisites should be fulfilled to configure the JRun application server to work with eG agent. The following sections describe how to configure a JRun application server to work with eG agent.

2.4 Configuring a JRun application server 3.0

In order to configure a JRun application server for monitoring by an eG agent, you need to enable the monitoring logs and configure the **global.properties** for customizing the output of the monitoring logs, by following the steps given below:

1. Open the **global.properties** file that resides in the <JRUN_INSTALL_DIR>/lib directory.
2. Look for the entry **logging.loglevel** and alter its value to:

   ```
   logging.loglevel=info,warning,error,metrics
   ```

   This ensures that all information, warnings, errors and metrics are logged by the JRun server. The eG agent uses the logged metrics for collecting statistics related to a JRun server.

3. The eG agent requires the metrics reported by each JRun server instance to be logged to a separate file. To do so, set the **logging.filelogwriter.filename** property as shown below:

   ```
   logging.filelogwriter.filename={jrun.rootdir}/logs/{jrun.server.name}-{log.level}.log
   ```

   The above process creates a log file named <SERVER_NAME>-metrics.log, to which the measures generated during monitoring are logged. The eG agent reads the data available in this log file and displays the same in the monitor interface.

4. Next, set the value of the properties **logging.filelogwriter.rotationsize** and **logging.filelogwriter.rotationfiles** in the **global.properties** file to 100000 and 5 respectively, as shown below:

   ```
   logging.filelogwriter.rotationsize=100000
   logging.filelogwriter.rotationfiles=5
   ```

   The rotation size specifies the maximum size of a log file. The smaller the value, the lower the monitoring overhead.

5. Ensure that the **monitor.format** property is set to **{monitor.combined-format}**. Then, reconfigure the value of **monitor.combined-format** as shown below, to define the pattern in which the metrics are to be logged.

   ```
   monitor.combined-format=(jcp+web)  {{jcp.busyTh}+{web.busyTh}} {{jcp.delayTh}+
   {web.delayTh}} {{jcp.totalTh}+{web.totalTh}} {totalMemory} {freeMemory} {sessions}
   {sessionsInMem} {{jcp.handledRq}+{web.handledRq}} {{jcp.handledMs}+{web.handledMs}}
   {{jcp.bytesIn}+{web.bytesIn}} {{jcp.bytesOut}+{web.bytesOut}} {{jcp.delayMs}+
   {web.delayMs}} {{jcp.delayRq}+{web.delayRq}} {{jcp.droppedRq}+{web.droppedRq}}
   ```

6. Then, set **monitor.interval** to a value (in seconds), which could be 80-90% of the test frequency. This value has to be very carefully set as all the measures that the agent reports would be relative to the frequency at which the logging is taking place.

   For example, if the eG agent executes all the related JRun tests at a frequency of 300 seconds, set the value of **monitor.interval** to 270 (300*0.09) as shown below:

   ```
   monitor.interval=270
   ```

7. Finally, restart the **JRun default server** service and check whether the metrics that have logged into the **default-metrics.log** file are in the following format:

   ```
   05/29 19:05:39 metrics (JRun) (jcp+web)  0 0 11 1984 747 0 0 25 340 6950 6852 0 0
   05/29 19:05:49 metrics (JRun) (jcp+web)  0 0 11 1984 606 0 0 14 171 3892 3836 0 0 0
   05/29 19:05:59 metrics (JRun) (jcp+web)  0 0 11 1984 523 0 0 0 0 0 0 0 0 0
   05/29 19:06:09 metrics (JRun) (jcp+web)  0 0 11 1984 440 0 0 0 0 0 0 0 0 0
   05/29 19:06:19 metrics (JRun) (jcp+web)  0 0 11 1984 358 0 0 0 0 0 0 0 0 0
   ```

2.5 Configuring a JRun Application Server 4.0

A single JRun server 4.0 may contain multiple server instances. The JRun application server 4.0 maintains individual logs for each of its server instances.

To monitor a specific server, modify the file **jrun.xml** which lies in the <JRUN_INSTALL_ DIR>/servers/<SERVER_NAME>/SERVER-INF directory, where JRUN_INSTALL_DIR is the home directory for the JRun server and SERVER_NAME is the name of the server instance. For example, if you have a server instance named APPLICATION 1, then the SERVER_NAME will be APPLICATION 1.

Make the following changes in this file:

1. Enable metrics logging by setting the **metricsEnabled** attribute of **LoggerService** to true as shown below.

   ```
   <attribute name="metricsEnabled">true</attribute>
   ```

2. Next, specify the file name to which the metrics must be logged. For that, add **{log.level}** to the **filename** attribute of the **FileLogEventHandler** service, as shown by the following example:

   ```
   <service class="jrunx.logger.FileLogEventHandler" name="FileLogEventHandler">
   <attribute name="filename">{jrun.rootdir}/logs/{jrun.server.name}-
   {log.level}.log</attribute>
   </service>
   ```

The above process creates a log file named <SERVER_NAME>-metrics.log, to which the measures generated during monitoring are logged. The eG agent reads the data available in this log file and displays the same in the monitor interface.

3. Set the **rotationSize** and **rotationFiles** attribute of the **FileLogEventHandler** service to 100k and 3 respectively as shown below:

```
<attribute name="rotationSize">100k</attribute>
<attribute name="rotationFiles">3</attribute>
```

The rotation size specifies the maximum size of a log file. The smaller the value, the lower the monitoring overhead.

4. To define the format in which metrics are to be logged, make the following changes in the **metricsFormat** attribute of the **LoggerService**.

```
<attribute name="metricsFormat">{{jrpp.busyTh}+{web.busyTh}} {{jrpp.delayTh}+
{web.delayTh}} {{jrpp.totalTh}+{web.totalTh}} {totalMemory} {freeMemory} {sessions}
{sessionsInMem} {{jrpp.handledRq}+{web.handledRq}} {{jrpp.handledMs}+{web.handledMs}}
{{jrpp.bytesIn}+{web.bytesIn}} {{jrpp.bytesOut}+{web.bytesOut}} {{jrpp.delayMs}+
{web.delayMs}} {{jrpp.delayRq}+{web.delayRq}} {{jrpp.droppedRq}+
{web.droppedRq}}</attribute>
```

5. Then, set the **metricsLogFrequency** attribute of **LoggerService** to a value (in seconds), which could be 80-90% of the test frequency. This value has to be very carefully set as all the measures that the agent reports would be relative to the frequency at which the logging is taking place.

For example, if the eG agent executes all the related JRun tests at a frequency of 300 seconds, set the value of metricsLogFrequency to 270 (300*0.09) as shown below:

```
<attribute name="metricsLogFrequency">270</attribute>
```

6. Finally, restart the specific JRun server for which you have enabled metrics logging and check whether the metrics that have logged into the <SERVER_NAME>-metrics.log file are in the following format:

```
06/14 12:06:58 metrics 0 0 1 32576 29198 1 1 0 0 0 0 0 0 0
06/14 12:11:28 metrics 0 0 2 32576 28310 1 1 18 17646 4535 2845 0 0 0
06/14 12:15:58 metrics 0 0 2 32576 29918 1 1 1 151 308 254 0 0 0
06/14 12:20:28 metrics 0 0 1 32576 29892 1 1 0 0 0 0 0 0 0
```

2.6 Configuring the eG Agent to Collect JVM-related Metrics from the JRun Server 3.0

The **JVM** layer of the *JRun* monitoring model is mapped to tests that report critical statistics related to the JRun server's JVM. These statistics typically reveal the following:

- The count of classes loaded/unloaded (Java Classes test)
- JVM thread usage (**JVM Threads** test)
- CPU and memory usage of the JVM (**JVM Cpu Usage** test and **JVM Memory Usage** test)
- The effectiveness of the JVM's garbage collection activity (**JVM Garbage Collections** test)
- The uptime of the JVM (JVM Uptime test)
- Whether JMX is currently enabled/disabled on the target WebLogic server (**JMX Connection to JVM** test)
- The count and status of file descriptors (**JVM File Descriptors** test)

These tests connect to the JRE used by the JRun application server to pull out the above-mentioned metrics. For these tests to execute, you should configure the eG agent to connect to JRE and collect the required metrics, using either of the following methodologies:

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

Since both JMX and SNMP support are available for JRE 1.5 and above only, these tests **will work only if the JRun server being monitored uses JRE 1.5 and above**.

If you choose to use JMX for pulling out the desired metrics from the JRE, then the following broad steps need to be followed before configuring these tests:

1. First, determine whether the JMX requires no authentication at all, or requires authentication (but no security)
2. If JMX does not require authentication, follow the steps below:
   - Login to the target JRun server.
   - Edit the *management.properties* file that is used by the JRE of the target JRun server, and configure the following in it:
   - The JMX remote port

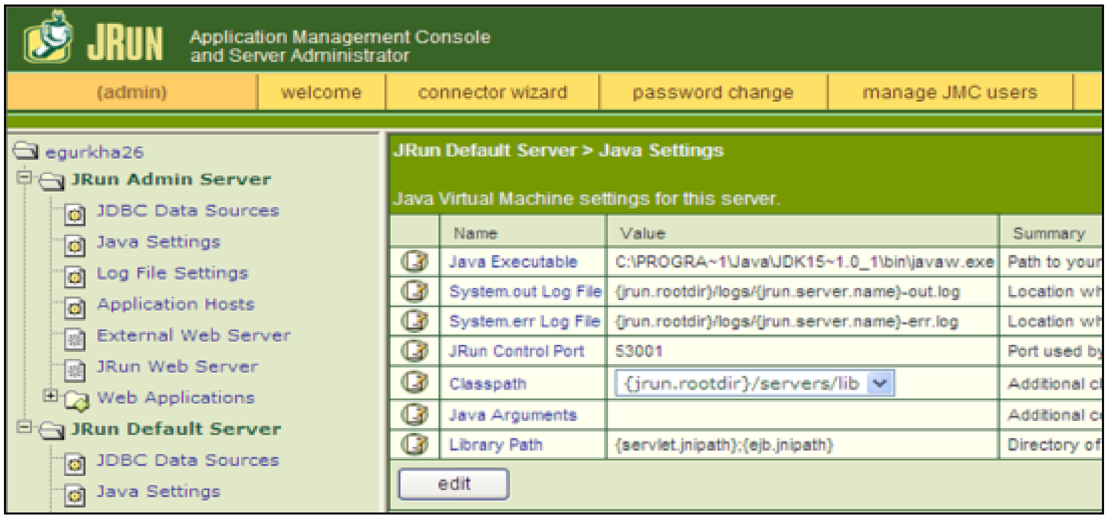
- Whether JMX is SSL-enabled or not
- Whether JMX requires authentication or not
- To know how to configure these, refer to the *Monitoring Java Applications* document.
- Save the file.

3. If the JMX requires authentication (but no security), follow the steps below:

- Login to the target JRun server. If the server is executing on a Windows host, then, login to the host as a local/domain administrator.
- Next, copy the *jmxremote.password.template* file in the <JAVA_HOME>\jre\lib\management folder 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.
- 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 *Monitoring Java Applications* document.
- Then, proceed to make the *jmxremote.password* file secure by granting a single user "full access" to that file. To know how to achieve this, refer to the *Monitoring Java Applications* document.
- Edit the *management.properties* file that is used by the JRE of the target JRun server, and configure the following in it:
    - The JMX remote port
    - Whether JMX is SSL-enabled or not
    - Whether JMX requires authentication or not
    - The full path to the *jmxremote.access* file
    - The full path to the *jmxremote.password* file

  To know how to configure these, refer to the *Monitoring Java Applications* document.
- Then, save the file.
- Next, connect to the JRun application management console using the URL: **http://<JRunServerIP>:<JRunPort>**
- The default port number of the JRun server is 8000.
- Provide the credentials for logging into the console, if prompted.
- When the console opens, you will find a tree-structure in the left panel of the console. Follow the node

sequence, *JRun Default Server -> Java Settings*, in the tree-structure.



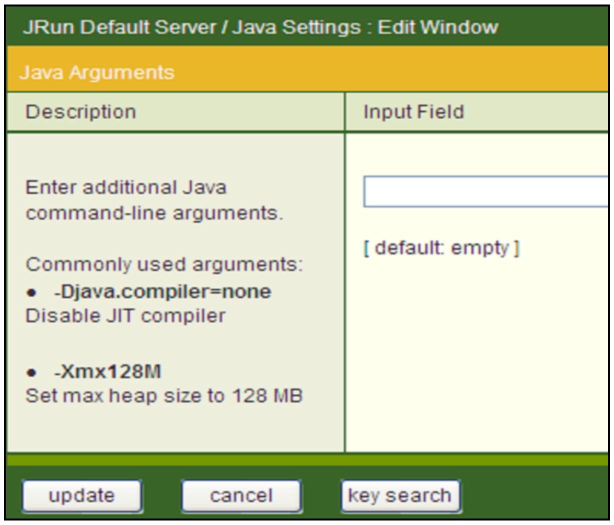
JRun Default Server > Java Settings

Java Virtual Machine settings for this server.

Name	Value	Summary
Java Executable	C:\PROGRA~1\Java\JDK15~1.0_1\bin\javaw.exe	Path to your
System.out Log File	{jrun.rootdir}/logs/{jrun.server.name}-out.log	Location wh
System.err Log File	{jrun.rootdir}/logs/{jrun.server.name}-err.log	Location wh
JRun Control Port	53001	Port used by
Classpath	{jrun.rootdir}/servers/lib	Additional cl
Java Arguments		Additional c
Library Path	{servlet.jnipath};{ejb.jnipath}	Directory of

Figure 2.3: Viewing the Java Virtual Machine Settings for the target JRun server

- The Java Virtual Machine settings will then appear in the right panel (see Figure 2.3). Click on the icon preceding the **Java Arguments** entry in the right panel.



JRun Default Server / Java Settings : Edit Window

Java Arguments

Description	Input Field
Enter additional Java command-line arguments. Commonly used arguments: • **-Djava.compiler=none** Disable JIT compiler • **-Xmx128M** Set max heap size to 128 MB	[default: empty]

Figure 2.4: Configuring the Java Arguments

- In the **Input Field** text box of Figure 2.4 that appears, enter the following:

```
-Dcom.sun.management.config.file=<Path to management.properties file used the JRE of
the target JRun server>
```

- Finally, stop and start the JRun server.

Note:

To know how to enable SNMP support for the JRE, refer to the *Monitoring Java Applications* document.

Once the above-said pre-requisistes are fulfilled, proceed to manage the JRun application server using the eG administrative interface. The steps for managing the server have been discussed in the below section.

Chapter 2: How to Monitor JRun Application Server Using eG Enterprise?

eG Enterprise monitors the JRun application servers in both agentless and agent-based manners. Before attempting JRun monitoring, a set of pre-requisites should be fulfilled to configure the JRun application server to work with eG agent. The following sections describe how to configure a JRun application server to work with eG agent.

2.7 Configuring a JRun application server 3.0

In order to configure a JRun application server for monitoring by an eG agent, you need to enable the monitoring logs and configure the **global.properties** for customizing the output of the monitoring logs, by following the steps given below:

1. Open the **global.properties** file that resides in the <JRUN_INSTALL_DIR>/lib directory.
2. Look for the entry **logging.loglevel** and alter its value to:

   ```
   logging.loglevel=info,warning,error,metrics
   ```

   This ensures that all information, warnings, errors and metrics are logged by the JRun server. The eG agent uses the logged metrics for collecting statistics related to a JRun server.

3. The eG agent requires the metrics reported by each JRun server instance to be logged to a separate file. To do so, set the **logging.filelogwriter.filename** property as shown below:

   ```
   logging.filelogwriter.filename={jrun.rootdir}/logs/{jrun.server.name}-{log.level}.log
   ```

   The above process creates a log file named <SERVER_NAME>-metrics.log, to which the measures generated during monitoring are logged. The eG agent reads the data available in this log file and displays the same in the monitor interface.

4. Next, set the value of the properties **logging.filelogwriter.rotationsize** and

logging.filelogwriter.rotationfiles in the **global.properties** file to 100000 and 5 respectively, as shown below:

```
logging.filelogwriter.rotationsize=100000
logging.filelogwriter.rotationfiles=5
```

The rotation size specifies the maximum size of a log file. The smaller the value, the lower the monitoring overhead.

5. Ensure that the **monitor.format** property is set to **{monitor.combined-format}**. Then, reconfigure the value of **monitor.combined-format** as shown below, to define the pattern in which the metrics are to be logged.

```
monitor.combined-format=(jcp+web)  {{jcp.busyTh}+{web.busyTh}} {{jcp.delayTh}+
{web.delayTh}} {{jcp.totalTh}+{web.totalTh}} {totalMemory} {freeMemory} {sessions}
{sessionsInMem} {{jcp.handledRq}+{web.handledRq}} {{jcp.handledMs}+{web.handledMs}}
{{jcp.bytesIn}+{web.bytesIn}} {{jcp.bytesOut}+{web.bytesOut}} {{jcp.delayMs}+
{web.delayMs}} {{jcp.delayRq}+{web.delayRq}} {{jcp.droppedRq}+{web.droppedRq}}
```

6. Then, set **monitor.interval** to a value (in seconds), which could be 80-90% of the test frequency. This value has to be very carefully set as all the measures that the agent reports would be relative to the frequency at which the logging is taking place.

   For example, if the eG agent executes all the related JRun tests at a frequency of 300 seconds, set the value of **monitor.interval** to 270 (300*0.09) as shown below:

```
monitor.interval=270
```

7. Finally, restart the **JRun default server** service and check whether the metrics that have logged into the **default-metrics.log** file are in the following format:

```
05/29 19:05:39 metrics (JRun) (jcp+web)  0 0 11 1984 747 0 0 25 340 6950 6852 0 0
05/29 19:05:49 metrics (JRun) (jcp+web)  0 0 11 1984 606 0 0 14 171 3892 3836 0 0 0
05/29 19:05:59 metrics (JRun) (jcp+web)  0 0 11 1984 523 0 0 0 0 0 0 0 0 0
05/29 19:06:09 metrics (JRun) (jcp+web)  0 0 11 1984 440 0 0 0 0 0 0 0 0 0
05/29 19:06:19 metrics (JRun) (jcp+web)  0 0 11 1984 358 0 0 0 0 0 0 0 0 0
```

2.8 Configuring a JRun Application Server 4.0

A single JRun server 4.0 may contain multiple server instances. The JRun application server 4.0 maintains individual logs for each of its server instances.

To monitor a specific server, modify the file **jrun.xml** which lies in the <JRUN_INSTALL_ DIR>/servers/<SERVER_NAME>/SERVER-INF directory, where JRUN_INSTALL_DIR is the home directory for the JRun server and SERVER_NAME is the name of the server instance. For example, if you have a server instance named APPLICATION 1, then the SERVER_NAME will be APPLICATION 1.

Make the following changes in this file:

1. Enable metrics logging by setting the **metricsEnabled** attribute of **LoggerService** to true as shown below.

```
<attribute name="metricsEnabled">true</attribute>
```

2. Next, specify the file name to which the metrics must be logged. For that, add **{log.level}** to the **filename** attribute of the **FileLogEventHandler** service, as shown by the following example:

```
<service class="jrunx.logger.FileLogEventHandler" name="FileLogEventHandler">
<attribute name="filename">{jrun.rootdir}/logs/{jrun.server.name}-
{log.level}.log</attribute>
</service>
```

   The above process creates a log file named <SERVER_NAME>-metrics.log, to which the measures generated during monitoring are logged. The eG agent reads the data available in this log file and displays the same in the monitor interface.

3. Set the **rotationSize** and **rotationFiles** attribute of the **FileLogEventHandler** service to 100k and 3 respectively as shown below:

```
<attribute name="rotationSize">100k</attribute>
<attribute name="rotationFiles">3</attribute>
```

   The rotation size specifies the maximum size of a log file. The smaller the value, the lower the monitoring overhead.

4. To define the format in which metrics are to be logged, make the following changes in the **metricsFormat** attribute of the **LoggerService**.

```
<attribute name="metricsFormat">{{jrpp.busyTh}+{web.busyTh}} {{jrpp.delayTh}+
{web.delayTh}} {{jrpp.totalTh}+{web.totalTh}} {totalMemory} {freeMemory} {sessions}
{sessionsInMem} {{jrpp.handledRq}+{web.handledRq}} {{jrpp.handledMs}+{web.handledMs}}
{{jrpp.bytesIn}+{web.bytesIn}} {{jrpp.bytesOut}+{web.bytesOut}} {{jrpp.delayMs}+
{web.delayMs}} {{jrpp.delayRq}+{web.delayRq}} {{jrpp.droppedRq}+
{web.droppedRq}}</attribute>
```

5. Then, set the **metricsLogFrequency** attribute of **LoggerService** to a value (in seconds), which could be 80-90% of the test frequency. This value has to be very carefully set as all the measures that the agent reports would be relative to the frequency at which the logging is taking place.

For example, if the eG agent executes all the related JRun tests at a frequency of 300 seconds, set the value of metricsLogFrequency to 270 (300*0.09) as shown below:

```
<attribute name="metricsLogFrequency">270</attribute>
```

6. Finally, restart the specific JRun server for which you have enabled metrics logging and check whether the metrics that have logged into the <SERVER_NAME>-metrics.log file are in the following format:

```
06/14 12:06:58 metrics 0 0 1 32576 29198 1 1 0 0 0 0 0 0 0
06/14 12:11:28 metrics 0 0 2 32576 28310 1 1 18 17646 4535 2845 0 0 0
06/14 12:15:58 metrics 0 0 2 32576 29918 1 1 1 151 308 254 0 0 0
06/14 12:20:28 metrics 0 0 1 32576 29892 1 1 0 0 0 0 0 0 0
```

2.9 Configuring the eG Agent to Collect JVM-related Metrics from the JRun Server 3.0

The **JVM** layer of the *JRun* monitoring model is mapped to tests that report critical statistics related to the JRun server's JVM. These statistics typically reveal the following:

- The count of classes loaded/unloaded (Java Classes test)
- JVM thread usage (**JVM Threads** test)
- CPU and memory usage of the JVM (**JVM Cpu Usage** test and **JVM Memory Usage** test)
- The effectiveness of the JVM's garbage collection activity (**JVM Garbage Collections** test)
- The uptime of the JVM (JVM Uptime test)
- Whether JMX is currently enabled/disabled on the target WebLogic server (**JMX Connection to JVM** test)
- The count and status of file descriptors (**JVM File Descriptors** test)

These tests connect to the JRE used by the JRun application server to pull out the above-mentioned metrics. For these tests to execute, you should configure the eG agent to connect to JRE and collect the required metrics, using either of the following methodologies:

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

Since both JMX and SNMP support are available for JRE 1.5 and above only, these tests **will work only if the JRun server being monitored uses JRE 1.5 and above**.

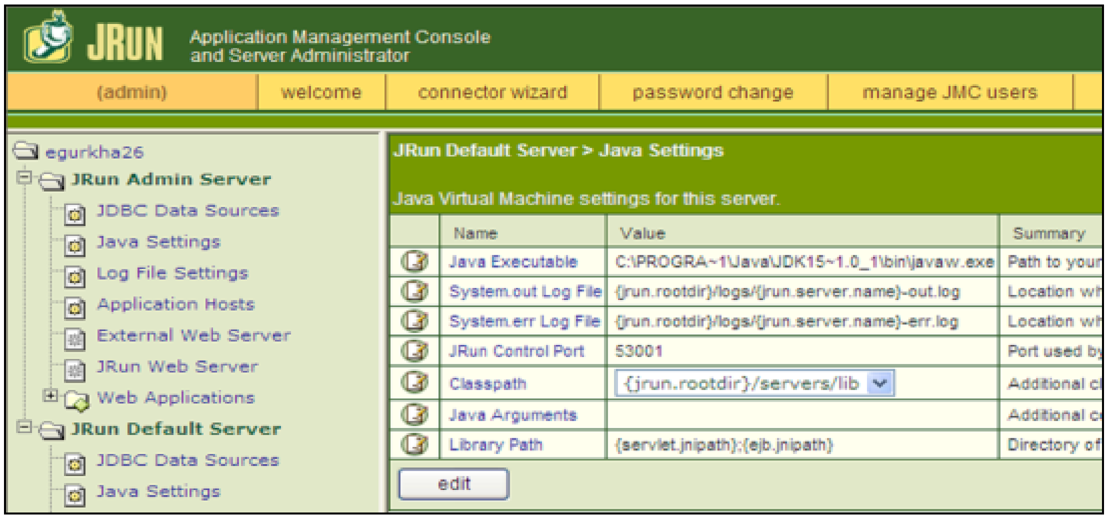
If you choose to use JMX for pulling out the desired metrics from the JRE, then the following broad steps need to be followed before configuring these tests:

1. First, determine whether the JMX requires no authentication at all, or requires authentication (but no security)
2. If JMX does not require authentication, follow the steps below:
   - Login to the target JRun server.
   - Edit the *management.properties* file that is used by the JRE of the target JRun server, and configure the following in it:
   - The JMX remote port
   - Whether JMX is SSL-enabled or not
   - Whether JMX requires authentication or not
   - To know how to configure these, refer to the *Monitoring Java Applications* document.
   - Save the file.
3. If the JMX requires authentication (but no security), follow the steps below:
   - Login to the target JRun server. If the server is executing on a Windows host, then, login to the host as a local/domain administrator.
   - Next, copy the *jmxremote.password.template* file in the <JAVA_HOME>\jre\lib\management folder 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.
   - 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 *Monitoring Java Applications* document.
   - Then, proceed to make the *jmxremote.password* file secure by granting a single user "full access" to that file. To know how to achieve this, refer to the *Monitoring Java Applications* document.
   - Edit the *management.properties* file that is used by the JRE of the target JRun server, and configure the following in it:
     - The JMX remote port
     - Whether JMX is SSL-enabled or not
     - Whether JMX requires authentication or not

- The full path to the *jmxremote.access* file
- The full path to the *jmxremote.password* file

To know how to configure these, refer to the *Monitoring Java Applications* document.

- Then, save the file.
- Next, connect to the JRun application management console using the URL: **http://<JRunServerIP>:<JRunPort>**
- The default port number of the JRun server is 8000.
- Provide the credentials for logging into the console, if prompted.
- When the console opens, you will find a tree-structure in the left panel of the console. Follow the node sequence, *JRun Default Server -> Java Settings*, in the tree-structure.



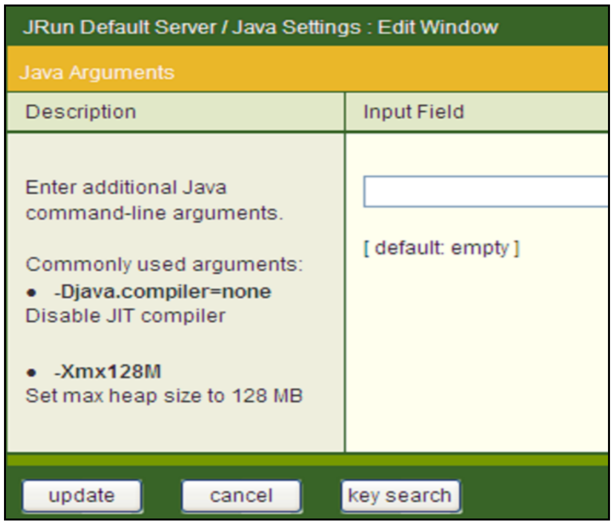
JRun Default Server > Java Settings

Java Virtual Machine settings for this server.

	Name	Value	Summary
	Java Executable	C:\PROGRA~1\Java\JDK15~1.0_1\bin\javaw.exe	Path to your
	System.out Log File	{jrun.rootdir}/logs/{jrun.server.name}-out.log	Location wh
	System.err Log File	{jrun.rootdir}/logs/{jrun.server.name}-err.log	Location wh
	JRun Control Port	53001	Port used by
	Classpath	{jrun.rootdir}/servers/lib	Additional cl
	Java Arguments		Additional c
	Library Path	{servlet.jnipath};{ejb.jnipath}	Directory of

Figure 2.5: Viewing the Java Virtual Machine Settings for the target JRun server

- The Java Virtual Machine settings will then appear in the right panel (see Figure 2.5). Click on the icon preceding the **Java Arguments** entry in the right panel.


JRun Default Server / Java Settings : Edit Window

Java Arguments

Description	Input Field
Enter additional Java command-line arguments. Commonly used arguments: • **-Djava.compiler=none** Disable JIT compiler • **-Xmx128M** Set max heap size to 128 MB	[default: empty]

Figure 2.6: Configuring the Java Arguments

- In the **Input Field** text box of Figure 2.6 that appears, enter the following:

```
-Dcom.sun.management.config.file=<Path to management.properties file used the JRE of
the target JRun server>
```

- Finally, stop and start the JRun server.

Note:

To know how to enable SNMP support for the JRE, refer to the *Monitoring Java Applications* document.

Once the above-said pre-requisistes are fulfilled, proceed to manage the JRun application server using the eG administrative interface. The steps for managing the server have been discussed in the below section.

Chapter 2: How to Monitor JRun Application Server Using eG Enterprise?

eG Enterprise monitors the JRun application servers in both agentless and agent-based manners. Before attempting JRun monitoring, a set of pre-requisites should be fulfilled to configure the JRun application server to work with eG agent. The following sections describe how to configure a JRun application server to work with eG agent.

2.10 Configuring a JRun application server 3.0

In order to configure a JRun application server for monitoring by an eG agent, you need to enable the monitoring logs and configure the **global.properties** for customizing the output of the monitoring logs, by following the steps given below:

1. Open the **global.properties** file that resides in the <JRUN_INSTALL_DIR>/lib directory.
2. Look for the entry **logging.loglevel** and alter its value to:

   ```
   logging.loglevel=info,warning,error,metrics
   ```

   This ensures that all information, warnings, errors and metrics are logged by the JRun server. The eG agent uses the logged metrics for collecting statistics related to a JRun server.

3. The eG agent requires the metrics reported by each JRun server instance to be logged to a separate file. To do so, set the **logging.filelogwriter.filename** property as shown below:

   ```
   logging.filelogwriter.filename={jrun.rootdir}/logs/{jrun.server.name}-{log.level}.log
   ```

   The above process creates a log file named <SERVER_NAME>-metrics.log, to which the measures generated during monitoring are logged. The eG agent reads the data available in this log file and displays the same in the monitor interface.

4. Next, set the value of the properties **logging.filelogwriter.rotationsize** and **logging.filelogwriter.rotationfiles** in the **global.properties** file to 100000 and 5 respectively, as shown below:

   ```
   logging.filelogwriter.rotationsize=100000
   logging.filelogwriter.rotationfiles=5
   ```

   The rotation size specifies the maximum size of a log file. The smaller the value, the lower the monitoring overhead.

5. Ensure that the **monitor.format** property is set to **{monitor.combined-format}**. Then, reconfigure the value of **monitor.combined-format** as shown below, to define the pattern in which the metrics are to be logged.

   ```
   monitor.combined-format=(jcp+web)  {{jcp.busyTh}+{web.busyTh}} {{jcp.delayTh}+
   {web.delayTh}} {{jcp.totalTh}+{web.totalTh}} {totalMemory} {freeMemory} {sessions}
   {sessionsInMem} {{jcp.handledRq}+{web.handledRq}} {{jcp.handledMs}+{web.handledMs}}
   {{jcp.bytesIn}+{web.bytesIn}} {{jcp.bytesOut}+{web.bytesOut}} {{jcp.delayMs}+
   {web.delayMs}} {{jcp.delayRq}+{web.delayRq}} {{jcp.droppedRq}+{web.droppedRq}}
   ```

6. Then, set **monitor.interval** to a value (in seconds), which could be 80-90% of the test frequency. This value has to be very carefully set as all the measures that the agent reports would be relative to the frequency at which the logging is taking place.

   For example, if the eG agent executes all the related JRun tests at a frequency of 300 seconds, set the value of **monitor.interval** to 270 (300*0.09) as shown below:

   ```
   monitor.interval=270
   ```

7. Finally, restart the **JRun default server** service and check whether the metrics that have logged into the **default-metrics.log** file are in the following format:

   ```
   05/29 19:05:39 metrics (JRun) (jcp+web)  0 0 11 1984 747 0 0 25 340 6950 6852 0 0
   05/29 19:05:49 metrics (JRun) (jcp+web)  0 0 11 1984 606 0 0 14 171 3892 3836 0 0 0
   05/29 19:05:59 metrics (JRun) (jcp+web)  0 0 11 1984 523 0 0 0 0 0 0 0 0 0
   05/29 19:06:09 metrics (JRun) (jcp+web)  0 0 11 1984 440 0 0 0 0 0 0 0 0 0
   05/29 19:06:19 metrics (JRun) (jcp+web)  0 0 11 1984 358 0 0 0 0 0 0 0 0 0
   ```

2.11 Configuring a JRun Application Server 4.0

A single JRun server 4.0 may contain multiple server instances. The JRun application server 4.0 maintains individual logs for each of its server instances.

To monitor a specific server, modify the file **jrun.xml** which lies in the <JRUN_INSTALL_ DIR>/servers/<SERVER_NAME>/SERVER-INF directory, where JRUN_INSTALL_DIR is the home directory for the JRun server and SERVER_NAME is the name of the server instance. For example, if you have a server instance named APPLICATION 1, then the SERVER_NAME will be APPLICATION 1.

Make the following changes in this file:

1. Enable metrics logging by setting the **metricsEnabled** attribute of **LoggerService** to true as shown below.

   ```
   <attribute name="metricsEnabled">true</attribute>
   ```

2. Next, specify the file name to which the metrics must be logged. For that, add **{log.level}** to the **filename** attribute of the **FileLogEventHandler** service, as shown by the following example:

   ```
   <service class="jrunx.logger.FileLogEventHandler" name="FileLogEventHandler">
   <attribute name="filename">{jrun.rootdir}/logs/{jrun.server.name}-
   {log.level}.log</attribute>
   </service>
   ```

The above process creates a log file named <SERVER_NAME>-metrics.log, to which the measures generated during monitoring are logged. The eG agent reads the data available in this log file and displays the same in the monitor interface.

3. Set the **rotationSize** and **rotationFiles** attribute of the **FileLogEventHandler** service to 100k and 3 respectively as shown below:

```
<attribute name="rotationSize">100k</attribute>
<attribute name="rotationFiles">3</attribute>
```

   The rotation size specifies the maximum size of a log file. The smaller the value, the lower the monitoring overhead.

4. To define the format in which metrics are to be logged, make the following changes in the **metricsFormat** attribute of the **LoggerService**.

```
<attribute name="metricsFormat">{{jrpp.busyTh}+{web.busyTh}} {{jrpp.delayTh}+
{web.delayTh}} {{jrpp.totalTh}+{web.totalTh}} {totalMemory} {freeMemory} {sessions}
{sessionsInMem} {{jrpp.handledRq}+{web.handledRq}} {{jrpp.handledMs}+{web.handledMs}}
{{jrpp.bytesIn}+{web.bytesIn}} {{jrpp.bytesOut}+{web.bytesOut}} {{jrpp.delayMs}+
{web.delayMs}} {{jrpp.delayRq}+{web.delayRq}} {{jrpp.droppedRq}+
{web.droppedRq}}</attribute>
```

5. Then, set the **metricsLogFrequency** attribute of **LoggerService** to a value (in seconds), which could be 80-90% of the test frequency. This value has to be very carefully set as all the measures that the agent reports would be relative to the frequency at which the logging is taking place.

   For example, if the eG agent executes all the related JRun tests at a frequency of 300 seconds, set the value of metricsLogFrequency to 270 (300*0.09) as shown below:

```
<attribute name="metricsLogFrequency">270</attribute>
```

6. Finally, restart the specific JRun server for which you have enabled metrics logging and check whether the metrics that have logged into the <SERVER_NAME>-metrics.log file are in the following format:

```
06/14 12:06:58 metrics 0 0 1 32576 29198 1 1 0 0 0 0 0 0 0
06/14 12:11:28 metrics 0 0 2 32576 28310 1 1 18 17646 4535 2845 0 0 0
06/14 12:15:58 metrics 0 0 2 32576 29918 1 1 1 151 308 254 0 0 0
06/14 12:20:28 metrics 0 0 1 32576 29892 1 1 0 0 0 0 0 0 0
```

2.12 Configuring the eG Agent to Collect JVM-related Metrics from the JRun Server 3.0

The **JVM** layer of the *JRun* monitoring model is mapped to tests that report critical statistics related to the JRun server's JVM. These statistics typically reveal the following:

- The count of classes loaded/unloaded (Java Classes test)
- JVM thread usage (**JVM Threads** test)
- CPU and memory usage of the JVM (**JVM Cpu Usage** test and **JVM Memory Usage** test)
- The effectiveness of the JVM's garbage collection activity (**JVM Garbage Collections** test)
- The uptime of the JVM (JVM Uptime test)
- Whether JMX is currently enabled/disabled on the target WebLogic server (**JMX Connection to JVM** test)
- The count and status of file descriptors (**JVM File Descriptors** test)

These tests connect to the JRE used by the JRun application server to pull out the above-mentioned metrics. For these tests to execute, you should configure the eG agent to connect to JRE and collect the required metrics, using either of the following methodologies:

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

Since both JMX and SNMP support are available for JRE 1.5 and above only, these tests **will work only if the JRun server being monitored uses JRE 1.5 and above**.

If you choose to use JMX for pulling out the desired metrics from the JRE, then the following broad steps need to be followed before configuring these tests:

1. First, determine whether the JMX requires no authentication at all, or requires authentication (but no security)
2. If JMX does not require authentication, follow the steps below:
    - Login to the target JRun server.
    - Edit the *management.properties* file that is used by the JRE of the target JRun server, and configure the following in it:
    - The JMX remote port

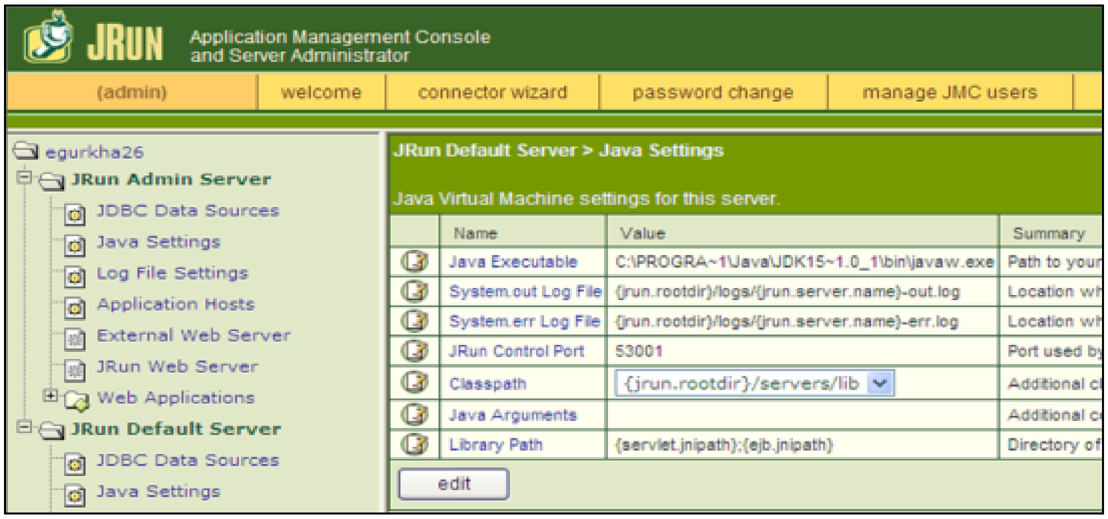
- Whether JMX is SSL-enabled or not
- Whether JMX requires authentication or not
- To know how to configure these, refer to the *Monitoring Java Applications* document.
- Save the file.

3. If the JMX requires authentication (but no security), follow the steps below:

   - Login to the target JRun server. If the server is executing on a Windows host, then, login to the host as a local/domain administrator.
   - Next, copy the *jmxremote.password.template* file in the <JAVA_HOME>\jre\lib\management folder 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.
   - 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 *Monitoring Java Applications* document.
   - Then, proceed to make the *jmxremote.password* file secure by granting a single user "full access" to that file. To know how to achieve this, refer to the *Monitoring Java Applications* document.
   - Edit the *management.properties* file that is used by the JRE of the target JRun server, and configure the following in it:
     - The JMX remote port
     - Whether JMX is SSL-enabled or not
     - Whether JMX requires authentication or not
     - The full path to the *jmxremote.access* file
     - The full path to the *jmxremote.password* file

     To know how to configure these, refer to the *Monitoring Java Applications* document.
   - Then, save the file.
   - Next, connect to the JRun application management console using the URL: **http://<JRunServerIP>:<JRunPort>**
   - The default port number of the JRun server is 8000.
   - Provide the credentials for logging into the console, if prompted.
   - When the console opens, you will find a tree-structure in the left panel of the console. Follow the node

sequence, *JRun Default Server -> Java Settings*, in the tree-structure.



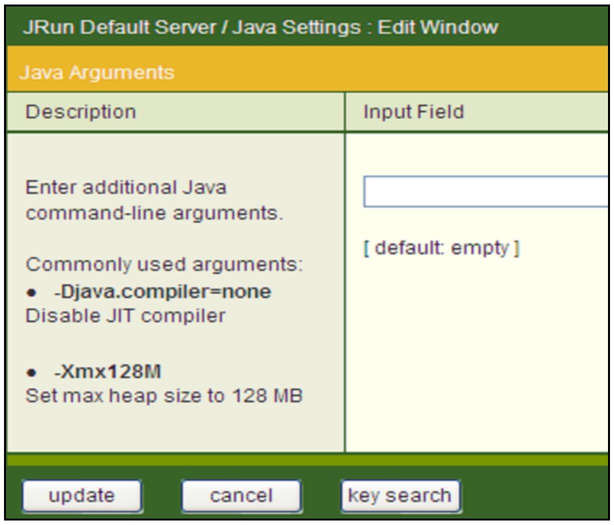
JRun Default Server > Java Settings

Java Virtual Machine settings for this server.

Name	Value	Summary
Java Executable	C:\PROGRA~1\Java\JDK15~1.0_1\bin\javaw.exe	Path to your
System.out Log File	{jrun.rootdir}/logs/{jrun.server.name}-out.log	Location wh
System.err Log File	{jrun.rootdir}/logs/{jrun.server.name}-err.log	Location wh
JRun Control Port	53001	Port used by
Classpath	{jrun.rootdir}/servers/lib	Additional cl
Java Arguments		Additional c
Library Path	{servlet.jnipath};{ejb.jnipath}	Directory of

Figure 2.7: Viewing the Java Virtual Machine Settings for the target JRun server

- The Java Virtual Machine settings will then appear in the right panel (see Figure 2.7). Click on the icon preceding the **Java Arguments** entry in the right panel.



JRun Default Server / Java Settings : Edit Window

Java Arguments

Description	Input Field
Enter additional Java command-line arguments. Commonly used arguments: • **-Djava.compiler=none** Disable JIT compiler • **-Xmx128M** Set max heap size to 128 MB	[default: empty]

Figure 2.8: Configuring the Java Arguments

- In the **Input Field** text box of Figure 2.8 that appears, enter the following:

```
-Dcom.sun.management.config.file=<Path to management.properties file used the JRE of
the target JRun server>
```

- Finally, stop and start the JRun server.

Note:

To know how to enable SNMP support for the JRE, refer to the *Monitoring Java Applications* document.

Once the above-said pre-requisistes are fulfilled, proceed to manage the JRun application server using the eG administrative interface. The steps for managing the server have been discussed in the below section.

2.13 Managing the JRun Application Server

The eG Enterprise can automatically discover the JRun application server. However, the discovered JRun component is managed manually. To achieve this, do the following:

1. Login to the eG administrative interface.
2. To manage the JRun server that is already discovered, directly proceed towards managing it using the **COMPONENTS – MANAGE/UNMANAGE** page (Infrastructure -> Components -> Manage/Unmanage).
3. However, if it is yet to be discovered, then run discovery (Infrastructure -> Components -> Discover) to get it discovered or add the component manually using the **COMPONENTS** page (Infrastructure -> Components -> Add/Modify). Remember that components manually added are managed automatically.
4. Discovered components, however, are managed using the **COMPONENTS – MANAGE / UNMANAGE** page. Figure 2.9 and Figure 2.10 clearly illustrate the process of managing the *JRun* server.

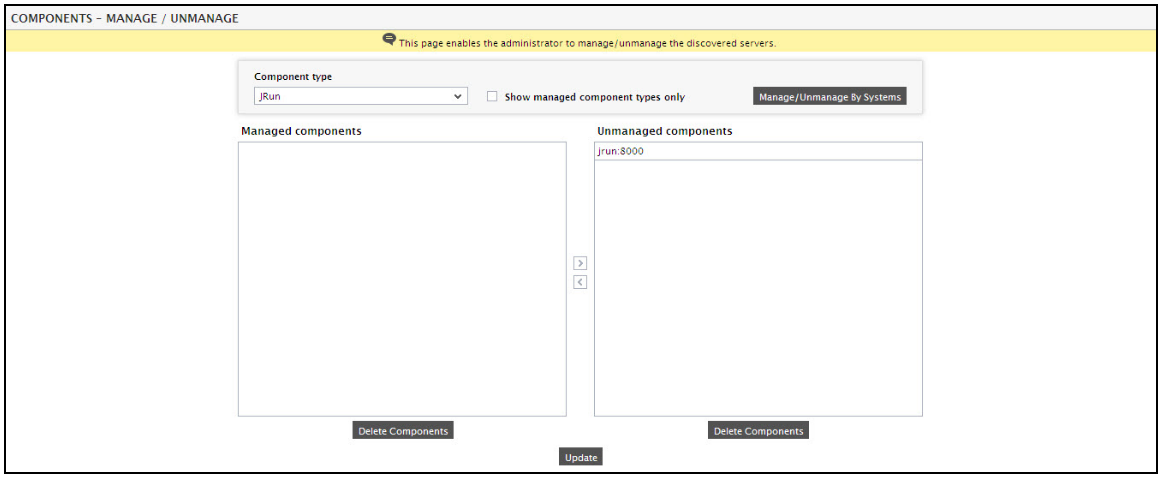
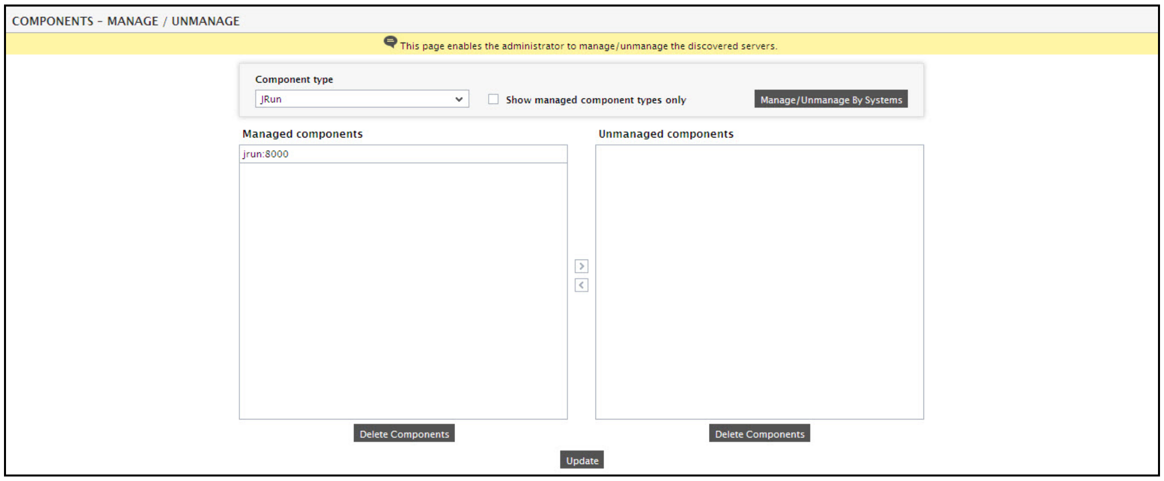

COMPONENTS - MANAGE / UNMANAGE
This page enables the administrator to manage/unmanage the discovered servers.
Component type
JRun
Show managed component types only
Manage/Unmanage By Systems
Managed components
Unmanaged components
jrun:8000
Delete Components
Delete Components
Update

Figure 2.9: Viewing the list of unmanaged JRun application servers


COMPONENTS - MANAGE / UNMANAGE

This page enables the administrator to manage/unmanage the discovered servers.

Component type: JRun

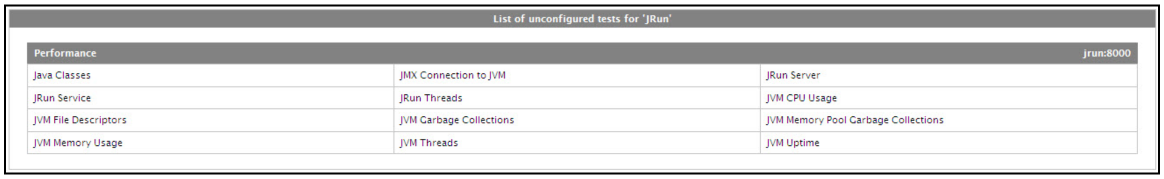
Show managed component types only

Managed components: jrun:8000

Unmanaged components

Figure 2.10: Managing the JRun application server

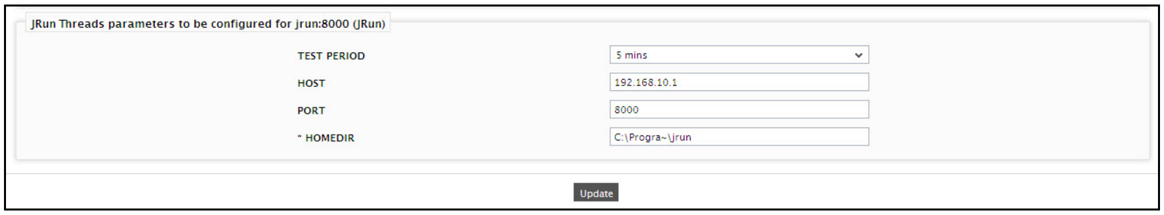
5. After managing the JRun application erver, on trying to sign out, you will be prompted to configure tests pertaining to the JRunServer (see Figure 2.11).


List of unconfigured tests for 'JRun'

Performance		jrun:8000
Java Classes	JMX Connection to JVM	JRun Server
JRun Service	JRun Threads	JVM CPU Usage
JVM File Descriptors	JVM Garbage Collections	JVM Memory Pool Garbage Collections
JVM Memory Usage	JVM Threads	JVM Uptime

Figure 2.11: A page displaying tests pertaining to the JRun application server

6. Click on any test in the list of unconfigured tests. For instance, click on the **JRun Threads** test to configure it. In the page that appears, specify the parameters as shown in Figure 2.12.


JRun Threads parameters to be configured for jrun:8000 (JRun)

TEST PERIOD	5 mins
HOST	192.168.10.1
PORT	8000
* HOMEDIR	C:\Progra~\jrun

Figure 2.12: Configuring the test parameters for the JRun

7. To know how to configure this test, refer to **Monitoring JRun Application Servers** chapter.
8. Finally, signout of the eG administrative interface.
9. The server is now all set to be monitored by the eG Enterprise Suite.

Chapter 3: Monitoring JRun Application Servers

eG Enterprise presents a hierarchical JRun monitoring model (see Figure 3.1), which executes diagnostic tests on the JRun server to extract a wide variety of performance heuristics. Using these statistics, administrators can guage the following:

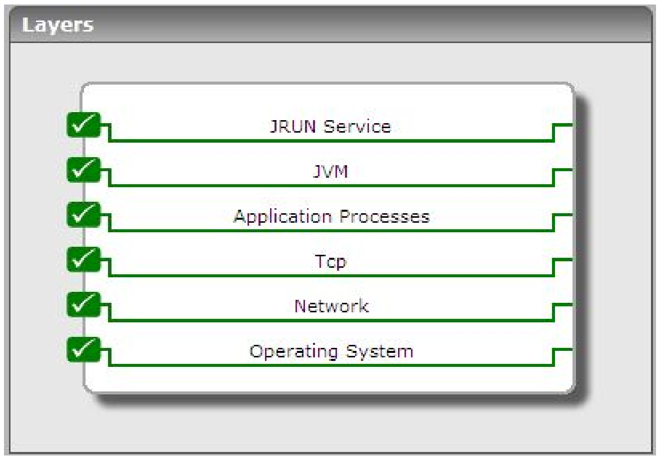


Figure 3.1: Layer model for a JRun application server

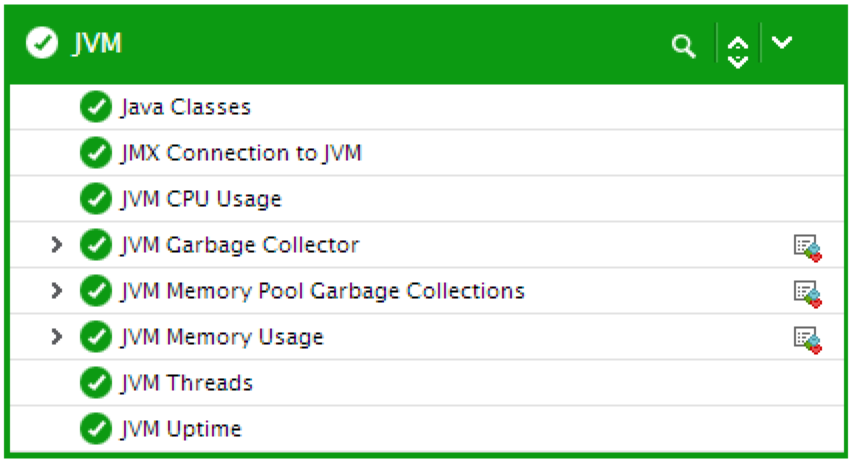
Each layer of the layer model is mapped to a series of tests that reports a wealth of performance metrics related to the appliance. These metrics can provide accurate answers for the following performance queries:

- Does the JRun server have adequate threads for handling its current and anticipated workload? Are too many requests waiting for threads?
- Is the server able to process requests quickly?
- Are there any requests for which processing has been delayed significantly?
- Have any requests been dropped by the server?
- Are too many requests awaiting processing?
- How quickly does the server respond to a request?
- Has sufficient memory been allocated to the server to serve requests effectively?
- Is the user activity on the server unusually high?

The sections to come will discuss the **JVM** and **JRUN Service** layers only, as the other layers have been discussed elaborately in the *Monitoring Unix and Windows Servers* document.

3.1 The JVM Layer

This layer collectively reports the resource usage and overall health of the JEUS JVM.


JVM

Java Classes

JMX Connection to JVM

JVM CPU Usage

JVM Garbage Collector

JVM Memory Pool Garbage Collections

JVM Memory Usage

JVM Threads

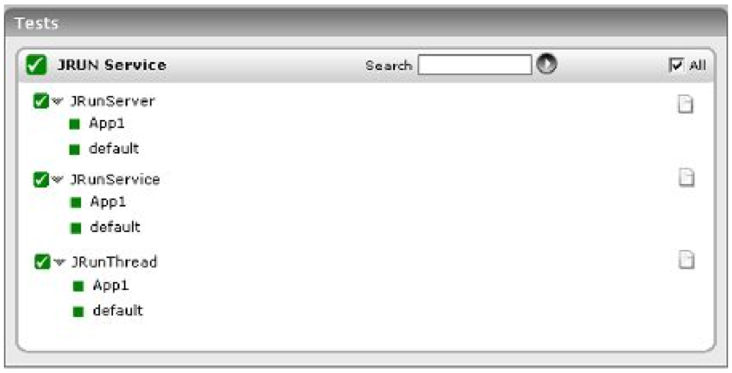
JVM Uptime

Figure 3.2: The tests mapped to the JVM layer

All the tests listed in Figure 3.2 have been discussed in the *Monitoring Java Applications* document.

3.2 The JRUN Service Layer

This layer tracks the health of the JRun service. The status of the **JRUN Service** layer is determined by the results of three tests, namely: JRunThread test, JRunService test and JRunServer test.



JRUN Service

JRunServer
- App1
- default

JRunService
- App1
- default

JRunThread
- App1
- default

Figure 3.3: Tests mapping to the JRUN Service layer

3.2.1 JRun Threads Test

The JRun server includes a thread pool, wherein each thread in the pool is used to service an incoming request. The size of the thread pool is dynamically increased / decreased depending on the rate of incoming requests. Since a single JRun server may comprise of multiple instances, the **JRun**

Thread test monitors the thread pool usage for each of the server instances. One set of results is reported per instance of a JRun server. In Figure 3.3, there are two instances of the JRun server. Hence, two sets of results are reported corresponding to the default, App1 instances.

Target of the test : An Allaire JRun Server

Agent deploying the test : An internal agent

Outputs of the test : One set of results for the JRun Server being monitored.

Configurable parameters for the test

Parameter	Description
Test Period	How often should the test be executed.
Host	Specify the host name of the server for which the test is to be configured.
Port	The port number on which the host is listening.
HomeDir	The directory in which the JRun server has been installed. In Windows environments, if the JRun server is installed at *D:\program files\allaire\jrun*, HomeDir has to be set to *D:\progra~1\allaire\jrun*.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Thread utilization	This indicates the percentage of existing threads, which are currently active.	Percent	A value consistently close to 100% is indicative of a bottleneck in the JRun server. In this case, incoming requests may have to wait for threads to be available to process the requests and the user may see an increased delay while accessing the server. To improve the performance of a JRun server 3.0, consider increasing the jcp.endpoint.main.active.threads property in the local.properties configuration file (residing in the JRUN_HOME_ DIR>/servers/servername), so that the server can allocate more threads for processing user requests.

Measurement	Description	Measurement Unit	Interpretation
			In case of a JRun server 4.0 modify the activeHandlerThreads attribute of the WebService in the jrun.xml file located in the <JRUN_HOME_ DIR>/servers/<SERVER_ NAME>/SERVER-INF directory. If the value of this metric is below 20%, the thread pool size may be set to be too large. Hence, consider reducing the jcp.endpoint.main.active.threads property in local.properties file of the JRun server 3.0. In case of a JRun server 4.0, modify the activeHandlerThreads attribute of the WebService in the jrun.xml file located in the <JRUN_HOME_ DIR>/servers/<SERVER_ NAME>/SERVER-INF directory.
Waiting threads	This indicates the number threads that are being queued in the server.	Number	A high value indicates that several threads are getting queued since the server is busy processing other threads. To reduce this number for a JRun server 3.0, you should increase the value of the jcp.endpoint.main.active.threads property in the local.properties configuration file, so that the server's JVM can handle more threads simultaneously. At the same time, care must be taken to ensure that the server does not become less efficient. In case of a JRun server 4.0, modify the activeHandlerThreads attribute of the WebService in the jrun.xml file located in the <JRUN_HOME_ DIR>/servers/<SERVER_ NAME>/SERVER-INF directory.
Total threads	This metric indicates the overall number of	Number	This metric must be considered in conjunction with the Thread utilization

Measurement	Description	Measurement Unit	Interpretation
	threads in all states in the server.		metric. Where possible ensure that the number of total threads is not set to be much larger than the maximum number of simultaneous requests expected for the server.

Note:

If a new server instance is added to a JRun application server while its being monitored, it will take at least an hour for the new instance to appear in the eG monitor console.

3.2.2 JRun Service Test

This test monitors the processing of requests by a JRun server.

Target of the test : An Allaire JRun Server

Agent deploying the test : An internal agent

Outputs of the test : One set of results for the JRun Server being monitored.

Configurable parameters for the test

Parameter	Description
Test Period	How often should the test be executed.
Host	Specify the host name of the server for which the test is to be configured.
Port	The port number on which the host is listening.
HomeDir	The directory in which the JRun server has been installed. In Windows environments, if the JRun server is installed at *D:\program files\allaire\jrun*, HomeDir has to be set to *D:\progra~1\allaire\jrun*.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Avg queue time	This value indicates the average time a request	Secs	A low value here would indicate that the server is performing optimally. An

Measurement	Description	Measurement Unit	Interpretation
	spends waiting for processing by the server.		increase in queuing delay reflects a server bottleneck. For a JRun server 3.0, consider increasing the jcp.endpoint.main.active.threads property in the local.properties file, to ensure that additional threads are available to process incoming requests. In case of a JRun server 4.0, modify the activeHandlerThreads attribute of the WebService in the jrun.xml file located in the <JRUN_HOME_ DIR>/servers/<SERVER_ NAME>/SERVER-INF directory. A number of compute intensive tasks can also end up increasing the queuing time for a request.
Avg processing time	This value indicates the average time taken by the server to process a request.	Secs	A high value here would indicate that processing time per request is very high, as the requests might be performing complex/time-consuming computation. Consider determining which of the server side processing tasks is compute intensive and explore ways of minimizing the server processing overhead.
Avg response time	This indicates the average time spent by a request in queuing and processing.	Secs	An increase in response time can occur because there are too many simultaneous requests or because of a bottleneck with any of the applications executing on the server.
Data read rate	This indicates the total rate of data received by the server for all incoming requests.	KB/Sec	An unusually high value indicates an increase in workload to the server. The Avg queue time metric indicates whether the increased incoming data rate is impacting the server performance.
Data write rate	This indicates the total rate of data sent by the server in response to all	KB/Sec	An unusually high value indicates an increase in workload to the server. The Avg queue time metric indicates whether

Measurement	Description	Measurement Unit	Interpretation
	incoming requests.		the increased incoming data rate is impacting the server performance.
Delayed requests	This value indicates the number of requests delayed at the server.	Number	While the Avg queue time gives an indicator of how long requests are being queued, this measure provides an indicator of how many requests were queued at the server over the last measurement period. This value should be close to 0 for peak performance.
Dropped requests	This indicates the number of requests that were dropped since the server could not process them or queue them.	Number	This value should be close to 0 at most times. Therefore, if a JRun server 3.0 drops a large number of requests, consider increasing the parameter jcp.endpoint.main.max.threads, in the local.properties file. This parameter indicates the maximum number of requests that can be processed or queued at any instant. In case of a JRun server 4.0, modify the activeHandlerThreads attribute of the WebService in the jrun.xml file located in the <JRUN_HOME_ DIR>/servers/<SERVER_ NAME>/SERVER-INF directory.
Request rate	This indicates the rate of requests handled by the JRun server.	Reqs/sec	This value is an indicator of server workload across all its connectors.

Note:

If a new server instance is added to a JRun application server while its being monitored, it will take at least hour for the new instance to appear in the eG monitor console.

3.2.3 JRun Server Test

This test periodically tracks the measures related to the Java virtual machine of all the instances of a JRun server.

Target of the test : An Allaire JRun Server

Agent deploying the test : An internal agent

Outputs of the test : One set of results for the JRun Server being monitored.

Configurable parameters for the test

Parameter	Description
Test Period	How often should the test be executed.
Host	Specify the host name of the server for which the test is to be configured.
Port	The port number on which the host is listening.
HomeDir	The directory in which the JRun server has been installed. In Windows environments, if the JRun server is installed at *D:\program files\allaire\jrun*, HomeDir has to be set to *D:\progra~1\allaire\jrun*.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Total server memory	This value indicates the total amount of memory that is being currently used by the server.	MB	An unusually large usage of memory by the server is a cause for concern. Further analysis is required to determine whether any of the applications running have memory leaks. If the memory is still low, consider starting the server with a higher memory usage setting.
Free server memory	This value indicates the total amount of memory that is not in use by the server.	MB	A very low value of free memory is also an indication of high memory utilization within the JVM.
Active sessions	This indicates the current number of active sessions in the server.	Number	A high value of this measure may indicate an increase in the server workload.
Sessions in memory	This indicates the number of active sessions in the server memory. Sessions can be persisted to a session repository	Number	A high value may indicate that there are a large number of sessions residing in the memory. Some of these sessions might be active or inactive. If the value is too high, consider changing the value

Measurement	Description	Measurement Unit	Interpretation
	depending upon your JRun configuration.		of session.persistence to false. This would stop persistence of a session when the server is terminated.

Note:

If a new server instance is added to a JRun application server while its being monitored, it will take at least hour for the new instance to appear in the eG monitor console.

3.2.4 Web Service Test

A web service is a collection of open protocols and standards used for exchanging data between applications or systems. Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer. A complete web service is, therefore, any service that:

- Is available over the Internet or private (intranet) networks
- Uses a standardized XML messaging system
- Is not tied to any one operating system or programming language
- Is self-describing via a common XML grammar
- Is discoverable via a simple find mechanism

The basic web services platform is XML + HTTP. All the standard web services work using the following components:

- SOAP (Simple Object Access Protocol)
- UDDI (Universal Description, Discovery and Integration)
- WSDL (Web Services Description Language)

A web service enables communication among various applications by using open standards such as HTML, XML, WSDL, and SOAP. A web service takes the help of the following:

- XML to tag the data
- SOAP to transfer a message
- WSDL to describe the availability of service.

The following are the major uses of the Web Services:

- **Reusable application-components**: Often applications need repeated access to application-components like currency conversion, weather reports, or even language translation. In such cases, the web services can be used to offer the application-components as services with ease.
- **Connect existing software**: Web services can help to solve the interoperability problem by giving different applications a way to link their data. With Web services you can exchange data between different applications and different platforms. Any application can have a Web Service component. Web Services can be created regardless of programming language.

In certain environments, administrators are required to keep an eye on the web services that offer repeated access to the application-components i.e., operations so that the work load on the users using those applicaiton components can be minimized. If for some reason the web service takes too long to respond or is unavailable to cater to the needs of the users, then the users will be deprived of access to the application- components involved in that particular web service. To avoid such inconvenience caused to the users, administrators are required to continuously monitor the web services. The **Web Service** test helps administrators to perform this task perfectly. By continuosly monitoring each operation i.e., application component of a web service that is offered, using the SOAP commands, this test helps administrators identify the availability, response time and response code of the web service and quickly figure out discrepancies if any web service is deemed unavailable. This way, the web services can be kept available round the clock thus helping the users perform their tasks without any difficulty.

Target of the test : An Allaire JRun Server

Agent deploying the test : An internal agent

Outputs of the test : One set of results for each *WebService:Operation* i.e., application-component performed on the target server that is being monitored.

Configurable parameters for the test

Parameter	Description
Test Period	How often should the test be executed.
Host	Specify the host name of the server for which the test is to be configured.
Port	The port number on which the host is listening.
WSDL URL	This test emulates a user accessing a specific web service(s) on the target server to determine the availability and responsiveness of the server. to enable this emulation,

Parameter	Description
	you need to configure the test with the url of the web service that it should access. specify this url against the WSDL URL parameter. if required, you can even configure multiple WSDL URLs - one each for every web service that the test should attempt to access. if each WSDL URL configured requires special permissions for logging in, then, you need to configure the test with separate credentials for logging into every WSDL URL. likewise, you need to provide instructions to the test on how to validate the content returned by every WSDL URL, and also set an encoding format for each wsdl url. to enable administrators to easily configure the above per WSDL URL, eg enterprise provides a special interface. to access this interface, click on the encircled '+' button alongside the url text box in the test configuration page. alternatively, you can even click on the encircled '+' button adjacent to the WSDL URL parameter in the test configuration page. to know how to use this special interface, refer to section Section **3.2.4.1**.
Operations	Once the WSDL URL(s) are specified, the operations that are offered by the web services and those that are to be monitored have to be configured. To select the required operations for monitoring, eG Enterprise provides a special interface. To access this interface, click on the encircled '+' button alongside the Operations text box in the test configuration page. Alternatively, you can even click on the encircled '+' button adjacent to the Operations parameter in the test configuration page. To know how to use this special interface, refer to section Section **3.2.4.2**.
Timeout	Specify the duration (in seconds) for which this test should wait for a response from the server. If there is no response from the server beyond the configured duration, the test will timeout. By default, this is set to 30 seconds.
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
WSDL url availability	Indicates whether the web service was able to respond successfully to the query made by the test.	Percent	Availability failures could be caused by several factors such as the web service process(es) being down, the web service being misconfigured, a network failure, etc. Temporary unavailability may also occur if the web service is overloaded. Availability is determined based on the response code returned by the service. A response code between 200 to 300 indicates that the service is available.
WSDL response time	Indicates the time taken by the eG agent to get the configured web service.	Secs	Response time being high denotes a problem. Poor response times may be due to the service being overloaded or misconfigured. If the URL accessed involves the generation of dynamic content by the service, backend problems (e.g., an overload at the application server or a database failure) can also result in an increase in response time.
Port status	Indicates whether/not the port of the web server is reachable.		The values reported by this measure and the corresponding numeric equivalents are listed in the table below: **Note:** By default, this measure reports the above-mentioned Measure Values to indicate whether the server has been rebooted or not. In the graph of this

Measure Val- ues	Numeric Val- ues
Yes	1
No	0

Measurement	Description	Measurement Unit	Interpretation
			measure however, the Measure Values are represented using the numeric equivalents only.
TCP connection availability	Indicates whether the test managed to establish a TCP connection to the server.	Percent	Failure to establish a TCP connection may imply that either the web server process is not up, or that the process is not operating correctly. In some cases of extreme overload, the failure to establish a TCP connection may be a transient condition. As the load subsides, the server may start functioning properly again.
TCP connect time	This measure quantifies the time for establishing a TCP connection to the web server host.	Secs	Typically, the TCP connection establishment must be very small (of the order of a few milliseconds). Since TCP connection establishment is handled at the OS-level, rather than by the application, an increase in this value signifies a system-level bottleneck on the host that supports the web server.
Server response time	Indicates the time period between when the connection was established and when the web server sent back a response header to the client.	Secs	While the total response time may depend on several factors, this measure is typically, a very good indicator of a server bottleneck (e.g., because all the available server threads or processes are in use).
Response code	The response code returned by the web server for the simulated request.	Number	A value between 200 and 300 indicates a good response. A 4xx value indicates a problem with the requested content (eg., page not found). A 5xx value indicates a server error.
Service availability	Indicates whether/not the web service is available.	Percent	A value of 100 indicates that the web service is available and a value of 0 indicates that the web service is not available.

Measurement	Description	Measurement Unit	Interpretation
Operation status	Indicates whether/not the configured operation is present in the web service.		This measure will not report metrics if the OPERATION parameter in the test configuration page is none in the test configuration page.
Operation Content length	Indicates the response code returned by the server for the simulated request.	Number	A value between 200 and 300 indicates a good response. A 4xx value indicates a problem with the requested content (e.g., page not found). A 5xx value indicates a server error. This measure will not report metrics if the OPERATION parameter in the test configuration page is none or if an invalid Value is specified or if the Value is not specified in the HTML View tab while configuring the operation for monitoring in the test configuration page.
Operation Content validity	This measure validates whether the operation was successful in executing the request made to it.	Percent	A value of 100% indicates that the content returned by the test is valid. A value of 0% indicates that the content may not be valid. This capability for content validation is especially important for multi-tier web applications. For example, a user may not be able to login to the web site but the server may reply back with a valid HTML page where in the error message, say, “Invalid Login” is reported. In this case, the availability will be 100 % (since we got a valid HTML response). If the test is configured such that the content parameter should exclude the string “Invalid Login”, in the above scenario content validity would have a value 0. This measure will not report metrics if the Operation parameter in the test configuration page is none or if an

Measurement	Description	Measurement Unit	Interpretation
			invalid Value is specified or if the Value is not specified in the HTML View tab while configuring the operation for monitoring in the test configuration page.
Operation execution time	Indicates the time taken to invoke the configured operation in the web service.	Secs	This measure will not report metrics if the Operation parameter in the test configuration page is none or if an invalid Value is specified or if the Value is not specified in the HTML View tab while configuring the operation for monitoring in the test configuration page.

3.2.4.1 Configuring Multiple WSDL URLs for Monitoring

In order to enable the eG agent to connect to multiple WSDL URLs and pull out the required metrics from them, the eG administrative interface provides a special page using which different WSDL URLs and their corresponding operations that need to be monitored can be specified. To configure the WSDL URLs, do the following:

Figure 3.4: Configuring the WebService test

1. Click on the encircled '+' button alongside the WSDL URL text box. Figure 3.5 will then appear.

Figure 3.5: The WebService URL Configuration page

2. Specify the following in Figure 3.5:
    - **Name:** Specify a unique name by which the WSDL URL you will be specifiying shortly will be referred to across the eG user interface. This is the name that will appear as the descriptor of this test.
    - **URL:** Enter the WSDL URL of the web service that this test should access.

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

3. Similarly, you can add multiple URL specifications by clicking the **Add More** button. To remove a WSDL URL specification, click on the encircled '-' button corresponding to it. To clear all **WSDL URL** specifications, click the **Clear** button. To update all the changes you made, click the **Update** button.
4. Once **Update** is clicked, you will return to the test configuration page as shown in Figure 3.4. The **WSDL URL** text box in the test configuration page will display just the **Name**s - i.e., the unique display names - that you may have configured for the multiple WSDL URLs, as a comma-separated list. To view the complete WSDL URL specification, click the encircled '+' button alongside the **WSDL URL** text box, once again.

3.2.4.2 Configuring Multiple Operations for Monitoring - WebServiceTest

By default, the **WebServiceTest** test will be configured with the WSDL URLs that offer the web services that are to be monitored. To configure the operations that are offered by the WSDL URLs, do the following:

1. Click on the encircled '+' button alongside the Operations text box as shown in Figure 3.4. Figure 3.6 will then appear.

Figure 3.6: Configuring the Web Service Operation

2. Specify the following in Figure 3.6:

   - **Manager/Agent for accessing WSDL URL:** Select the eG agent or the eG Manager that is authorized to access the configured WSDL URL from this list.
   - **WSDL URL:** Once the eG agent/eG Manager is chosen from the *Manager/Agent for accessing WSDL URL* list, this list will be populated automatically with all the WSDL URLs specified in the **WSDL URL** text box (See Figure 3.4). Select the **WSDL URL** of your choice from this list.
   - **Services:** The web services offered by the chosen WSDL URL will then be populated in this list. Select a service of your choice from this list.
     - The operations that are offered by the chosen service will then be populated in the DEFINED Operations list. To monitor a chosen operation, select the operation and click the < button. This will move the chosen operation to the **MONITORED OPERATIONS** list.
     - Click the **Configure** button to save the changes.

    - The eG agent uses SOAP requests to obtain the necessary metrics from the web service. Once the operation is configured, the XML View of the SOAP Request corresponding to the chosen operation will be generated and listed in the **XML View** tab. Likewise, the **HTML View** tab lists the **SOAP Parameter** that is passed to collect the required metrics for the chosen operation.
    - To obtain operation-level statistics, it is important to specify a valid value in the **VALUE** text box of the **HTML View** tab as shown in Figure 3.7. Each time the test is executed, this value will be provided as an input to the chosen operation.

Figure 3.7: Specifying the value for the chosen operation

- Click the **Save and Cofigure More** button to save the changes made.
- If you wish to verify if the **VALUE** specified in the **HTML View** tab is valid, then you can do so by clicking the **Send Request** button. Figure 3.7 will then appear. If the value specified in the **VALUE** text box is indeed valid, then the operation will be performed on the value and the result will be specified. For example, if your chosen operation is *FahrenheittoCelsius*, the **SOAP Parameter** is *Farenheit* and the value that you wish to convert is *100*, the result will be specified in the **WEB SERVICE RESPONSE** pop up window as below: *<FahrenheitToCelsiusResult>37.7777777777778</FahrenheitToCelsiusResult>*

Figure 3.8: The value that appears when the operation is performed successfully

- If you have specified an invalid value, then a message as follows will be displayed in the pop up window: *<FahrenheitToCelsiusResult>Error</FahrenheitToCelsiusResult>*

Figure 3.9: An Error appearing during value conversion

- If you do not specify a **VALUE** or specify an invalid value, operation-level statistics will not be collected by the eG agent and such metrics will not be available in the eG monitoring interface.

3. Similarly, you can configure multiple Operations by clicking the **Configure** button in Figure 3.6. To remove an operation, select the operation from the **MONITORED OPERATION** list and click the **>** button.

Once **Save and Configure More** button is clicked, you will return to the test configuration page (see Figure 3.6). The Operations text box in the test configuration page will display just the operations, as a comma- separated list. To view the complete operation specification, click the encircled '+' button alongside the Operations text box, once again.

3.2.5 Web Service Test

A web service is a collection of open protocols and standards used for exchanging data between applications or systems. Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer. A complete web service is, therefore, any service that:

- Is available over the Internet or private (intranet) networks
- Uses a standardized XML messaging system
- Is not tied to any one operating system or programming language
- Is self-describing via a common XML grammar
- Is discoverable via a simple find mechanism

The basic web services platform is XML + HTTP. All the standard web services work using the following components:

- SOAP (Simple Object Access Protocol)
- UDDI (Universal Description, Discovery and Integration)
- WSDL (Web Services Description Language)

A web service enables communication among various applications by using open standards such as HTML, XML, WSDL, and SOAP. A web service takes the help of the following:

- XML to tag the data
- SOAP to transfer a message
- WSDL to describe the availability of service.

The following are the major uses of the Web Services:

- **Reusable application-components**: Often applications need repeated access to application-components like currency conversion, weather reports, or even language translation. In such cases, the web services can be used to offer the application-components as services with ease.

- **Connect existing software**: Web services can help to solve the interoperability problem by giving different applications a way to link their data. With Web services you can exchange data between different applications and different platforms. Any application can have a Web Service component. Web Services can be created regardless of programming language.

In certain environments, administrators are required to keep an eye on the web services that offer repeated access to the application-components i.e., operations so that the work load on the users using those applicaiton components can be minimized. If for some reason the web service takes too long to respond or is unavailable to cater to the needs of the users, then the users will be deprived of access to the application-components involved in that particular web service. To avoid such inconvenience caused to the users, administrators are required to continuously monitor the web services. The **Web Service** test helps administrators to perform this task perfectly. By continuosly monitoring each operation i.e., application component of a web service that is offered, using the SOAP commands, this test helps administrators identify the availability, response time and response code of the web service and quickly figure out discrepancies if any web service is deemed unavailable. This way, the web services can be kept available round the clock thus helping the users perform their tasks without any difficulty.

Target of the test : An Allaire JRun Server

Agent deploying the test : An internal agent

Outputs of the test : One set of results for each *WebService:Operation* i.e., application-component performed on the target server that is being monitored.

Configurable parameters for the test

Parameter	Description
Test Period	How often should the test be executed.
Host	Specify the host name of the server for which the test is to be configured.
Port	The port number on which the host is listening.
WSDL URL	This test emulates a user accessing a specific web service(s) on the target server to determine the availability and responsiveness of the server. to enable this emulation, you need to configure the test with the url of the web service that it should access. specify this url against the WSDL URL parameter. if required, you can even configure multiple WSDL URLs - one each for every web service that the test should attempt to access. if each WSDL URL configured requires special permissions for logging in,

Parameter	Description
	then, you need to configure the test with separate credentials for logging into every WSDL URL. likewise, you need to provide instructions to the test on how to validate the content returned by every WSDL URL, and also set an encoding format for each wsdl url. to enable administrators to easily configure the above per WSDL URL, eg enterprise provides a special interface. to access this interface, click on the encircled '+' button alongside the url text box in the test configuration page. alternatively, you can even click on the encircled '+' button adjacent to the WSDL URL parameter in the test configuration page. to know how to use this special interface, refer to section Section **3.2.5.1**.
Operations	Once the WSDL URL(s) are specified, the operations that are offered by the web services and those that are to be monitored have to be configured. To select the required operations for monitoring, eG Enterprise provides a special interface. To access this interface, click on the encircled '+' button alongside the Operations text box in the test configuration page. Alternatively, you can even click on the encircled '+' button adjacent to the Operations parameter in the test configuration page. To know how to use this special interface, refer to section Section **3.2.5.2**.
Timeout	Specify the duration (in seconds) for which this test should wait for a response from the server. If there is no response from the server beyond the configured duration, the test will timeout. By default, this is set to 30 seconds.
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
WSDL url availability	Indicates whether the web service was able to	Percent	Availability failures could be caused by several factors such as the web

Measurement	Description	Measurement Unit	Interpretation
	respond successfully to the query made by the test.		service process(es) being down, the web service being misconfigured, a network failure, etc. Temporary unavailability may also occur if the web service is overloaded. Availability is determined based on the response code returned by the service. A response code between 200 to 300 indicates that the service is available.
WSDL response time	Indicates the time taken by the eG agent to get the configured web service.	Secs	Response time being high denotes a problem. Poor response times may be due to the service being overloaded or misconfigured. If the URL accessed involves the generation of dynamic content by the service, backend problems (e.g., an overload at the application server or a database failure) can also result in an increase in response time.
Port status	Indicates whether/not the port of the web server is reachable.		The values reported by this measure and the corresponding numeric equivalents are listed in the table below: **Note:** By default, this measure reports the above-mentioned Measure Values to indicate whether the server has been rebooted or not. In the graph of this measure however, the Measure Values are represented using the numeric equivalents only.

Measure Values	Numeric Values
Yes	1
No	0

Measurement	Description	Measurement Unit	Interpretation
TCP connection availability	Indicates whether the test managed to establish a TCP connection to the server.	Percent	Failure to establish a TCP connection may imply that either the web server process is not up, or that the process is not operating correctly. In some cases of extreme overload, the failure to establish a TCP connection may be a transient condition. As the load subsides, the server may start functioning properly again.
TCP connect time	This measure quantifies the time for establishing a TCP connection to the web server host.	Secs	Typically, the TCP connection establishment must be very small (of the order of a few milliseconds). Since TCP connection establishment is handled at the OS-level, rather than by the application, an increase in this value signifies a system-level bottleneck on the host that supports the web server.
Server response time	Indicates the time period between when the connection was established and when the web server sent back a response header to the client.	Secs	While the total response time may depend on several factors, this measure is typically, a very good indicator of a server bottleneck (e.g., because all the available server threads or processes are in use).
Response code	The response code returned by the web server for the simulated request.	Number	A value between 200 and 300 indicates a good response. A 4xx value indicates a problem with the requested content (eg., page not found). A 5xx value indicates a server error.
Service availability	Indicates whether/not the web service is available.	Percent	A value of 100 indicates that the web service is available and a value of 0 indicates that the web service is not available.
Operation status	Indicates whether/not the configured operation is present in the web		This measure will not report metrics if the OPERATION parameter in the test configuration page is none in the test

Measurement	Description	Measurement Unit	Interpretation
	service.		configuration page.
Operation Content length	Indicates the response code returned by the server for the simulated request.	Number	A value between 200 and 300 indicates a good response. A 4xx value indicates a problem with the requested content (e.g., page not found). A 5xx value indicates a server error. This measure will not report metrics if the OPERATION parameter in the test configuration page is none or if an invalid Value is specified or if the Value is not specified in the HTML View tab while configuring the operation for monitoring in the test configuration page.
Operation Content validity	This measure validates whether the operation was successful in executing the request made to it.	Percent	A value of 100% indicates that the content returned by the test is valid. A value of 0% indicates that the content may not be valid. This capability for content validation is especially important for multi-tier web applications. For example, a user may not be able to login to the web site but the server may reply back with a valid HTML page where in the error message, say, “Invalid Login” is reported. In this case, the availability will be 100 % (since we got a valid HTML response). If the test is configured such that the content parameter should exclude the string “Invalid Login”, in the above scenario content validity would have a value 0. This measure will not report metrics if the Operation parameter in the test configuration page is none or if an invalid Value is specified or if the Value is not specified in the HTML View tab while configuring the operation for

Measurement	Description	Measurement Unit	Interpretation
			monitoring in the test configuration page.
Operation execution time	Indicates the time taken to invoke the configured operation in the web service.	Secs	This measure will not report metrics if the Operation parameter in the test configuration page is none or if an invalid Value is specified or if the Value is not specified in the HTML View tab while configuring the operation for monitoring in the test configuration page.

3.2.5.1 Configuring Multiple WSDL URLs for Monitoring

In order to enable the eG agent to connect to multiple WSDL URLs and pull out the required metrics from them, the eG administrative interface provides a special page using which different WSDL URLs and their corresponding operations that need to be monitored can be specified. To configure the WSDL URLs, do the following:

Figure 3.10: Configuring the WebService test

1. Click on the encircled '+' button alongside the WSDL URL text box. Figure 3.11 will then appear.

Figure 3.11: The WebService URL Configuration page

2. Specify the following in Figure 3.11:
   - **Name:** Specify a unique name by which the WSDL URL you will be specifiying shortly will be referred to across the eG user interface. This is the name that will appear as the descriptor of this test.
   - **URL:** Enter the WSDL URL of the web service that this test should access.
   - **Username** and **Password:** These parameters are to be set only if a specific user name / password has to be specified to login to the web service (i.e., WSDL URL) that you have configured for monitoring. In this case, provide valid login credentials using the **Username** and

Password text boxes. If the server on which **WebService** test executes supports 'Anonymous user access', then these parameters will take either of the following values:

- A valid Username and Password for the configured WSDL URL
- *none* in both the **Username** and **Password** text boxes of the configured WSDL URL, if no user authorization is required
- Some servers however, support NTLM (Integrated Windows) authentication, where valid login credentials are mandatory. In other words, a *none* specification will not be supported by such servers. Therefore, in this case, against each configured WSDL URL, you will have to provide a valid **Username** in the format: *domainname\username*, followed by a valid **Password**.
- Please be sure to check if your web service requires HTTP authentication while configuring this parameter. HTTP authentication typically involves a separate pop-up window when you try to access the page. Many services use HTTP POST for obtaining the user name and password and validating the user login. In such cases, the username and password have to be provided as part of the POST information and NOT as part of the **CREDENTIALS** specification for the WebService test.

- **Content**: The **Content** parameter has to be configured with an instruction:value pair that will be used to validate the content being returned by the test. If the **Content** value is *None*, no validation is performed. On the other hand, if you pick the *Include* option from the **Content** list, it indicates to the test that for the content returned by the web server to be valid, the content must include the specified value (a simple string search is done in this case). This value should be specified in the adjacent text box. Similarly, if the *Exclude* option is chosen from the **Content** drop-down, it indicates to the test that the server's output is valid if it does not contain the value specified in the adjacent text box. The *Include* or *Exclude* value you specify in the text box can include wildcard characters. For example, an *Include* instruction can be **Home page**.
- **Encoding**: Sometimes the eG agent has to parse the **WSDL URL** content with specific encoding other than the default (ISO-8859-1) encoding. In such a case, specify the type of encoding using which the eG agent can parse the WSDL URL content in the **Encoding** text box. By default, this value is *none*.

3. Similarly, you can add multiple URL specifications by clicking the **Add More** button. To remove a WSDL URL specification, click on the encircled '-' button corresponding to it. To clear all **WSDL URL** specifications, click the **Clear** button. To update all the changes you made, click the **Update** button.

4. Once **Update** is clicked, you will return to the test configuration page as shown in Figure 3.10. The **WSDL URL** text box in the test configuration page will display just the **Name**s - i.e., the unique display names - that you may have configured for the multiple WSDL URLs, as a comma-separated list. To view the complete WSDL URL specification, click the encircled '+' button alongside the **WSDL URL** text box, once again.

3.2.5.2 Configuring Multiple Operations for Monitoring - WebServiceTest

By default, the **WebServiceTest** test will be configured with the WSDL URLs that offer the web services that are to be monitored. To configure the operations that are offered by the WSDL URLs, do the following:

1. Click on the encircled '+' button alongside the Operations text box as shown in Figure 3.10. Figure 3.12 will then appear.

Figure 3.12: Configuring the Web Service Operation

2. Specify the following in Figure 3.12:

   - **Manager/Agent for accessing WSDL URL:** Select the eG agent or the eG Manager that is authorized to access the configured WSDL URL from this list.
   - **WSDL URL:** Once the eG agent/eG Manager is chosen from the *Manager/Agent for accessing WSDL URL* list, this list will be populated automatically with all the WSDL URLs specified in the **WSDL URL** text box (See Figure 3.10). Select the **WSDL URL** of your choice from this list.
   - **Services:** The web services offered by the chosen WSDL URL will then be populated in this list. Select a service of your choice from this list.
     - The operations that are offered by the chosen service will then be populated in the DEFINED Operations list. To monitor a chosen operation, select the operation and click the < button. This will move the chosen operation to the **MONITORED OPERATIONS** list.
     - Click the **Configure** button to save the changes.
     - The eG agent uses SOAP requests to obtain the necessary metrics from the web service. Once the operation is configured, the XML View of the SOAP Request corresponding to the chosen operation will be generated and listed in the **XML View**

tab. Likewise, the **HTML View** tab lists the **SOAP Parameter** that is passed to collect the required metrics for the chosen operation.

- To obtain operation-level statistics, it is important to specify a valid value in the **VALUE** text box of the **HTML View** tab as shown in Figure 3.13. Each time the test is executed, this value will be provided as an input to the chosen operation.

Figure 3.13: Specifying the value for the chosen operation

- Click the **Save and Cofigure More** button to save the changes made.
- If you wish to verify if the **VALUE** specified in the **HTML View** tab is valid, then you can do so by clicking the **Send Request** button. Figure 3.13 will then appear. If the value specified in the **VALUE** text box is indeed valid, then the operation will be performed on the value and the result will be specified. For example, if your chosen operation is *FahrenheittoCelsius*, the **SOAP Parameter** is *Farenheit* and the value that you wish to convert is *100*, the result will be specified in the **WEB SERVICE RESPONSE** pop up window as below: *<FahrenheitToCelsiusResult>37.7777777777778</FahrenheitToCelsiusResult>*

Figure 3.14: The value that appears when the operation is performed successfully

- If you have specified an invalid value, then a message as follows will be displayed in the pop up window: *<FahrenheitToCelsiusResult>Error</FahrenheitToCelsiusResult>*

Figure 3.15: An Error appearing during value conversion

- If you do not specify a **VALUE** or specify an invalid value, operation-level statistics will not be collected by the eG agent and such metrics will not be available in the eG monitoring interface.

3. Similarly, you can configure multiple Operations by clicking the **Configure** button in Figure 3.12. To remove an operation, select the operation from the **MONITORED OPERATION** list and click the **>** button.

Once **Save and Configure More** button is clicked, you will return to the test configuration page (see Figure 3.12). The Operations text box in the test configuration page will display just the operations, as a comma-separated list. To view the complete operation specification, click the encircled '+' button alongside the Operations text box, once again.

3.2.6 Web Service Test

A web service is a collection of open protocols and standards used for exchanging data between applications or systems. Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer. A complete web service is, therefore, any service that:

- Is available over the Internet or private (intranet) networks
- Uses a standardized XML messaging system
- Is not tied to any one operating system or programming language
- Is self-describing via a common XML grammar
- Is discoverable via a simple find mechanism

The basic web services platform is XML + HTTP. All the standard web services work using the following components:

- SOAP (Simple Object Access Protocol)
- UDDI (Universal Description, Discovery and Integration)
- WSDL (Web Services Description Language)

A web service enables communication among various applications by using open standards such as HTML, XML, WSDL, and SOAP. A web service takes the help of the following:

- XML to tag the data
- SOAP to transfer a message
- WSDL to describe the availability of service.

The following are the major uses of the Web Services:

- **Reusable application-components**: Often applications need repeated access to application-components like currency conversion, weather reports, or even language translation. In such cases, the web services can be used to offer the application-components as services with ease.

- **Connect existing software**: Web services can help to solve the interoperability problem by giving different applications a way to link their data. With Web services you can exchange data between different applications and different platforms. Any application can have a Web Service component. Web Services can be created regardless of programming language.

In certain environments, administrators are required to keep an eye on the web services that offer repeated access to the application-components i.e., operations so that the work load on the users using those applicaiton components can be minimized. If for some reason the web service takes too long to respond or is unavailable to cater to the needs of the users, then the users will be deprived of access to the application- components involved in that particular web service. To avoid such inconvenience caused to the users, administrators are required to continuously monitor the web services. The **Web Service** test helps administrators to perform this task perfectly. By continuosly monitoring each operation i.e., application component of a web service that is offered, using the SOAP commands, this test helps administrators identify the availability, response time and response code of the web service and quickly figure out discrepancies if any web service is deemed unavailable. This way, the web services can be kept available round the clock thus helping the users perform their tasks without any difficulty.

Target of the test : An Allaire JRun Server

Agent deploying the test : An internal agent

Outputs of the test : One set of results for each *WebService:Operation* i.e., application-component performed on the target server that is being monitored.

Configurable parameters for the test

Parameter	Description
Test Period	How often should the test be executed.
Host	Specify the host name of the server for which the test is to be configured.
Port	The port number on which the host is listening.
WSDL URL	This test emulates a user accessing a specific web service(s) on the target server to determine the availability and responsiveness of the server. to enable this emulation, you need to configure the test with the url of the web service that it should access. specify this url against the WSDL URL parameter. if required, you can even configure multiple WSDL URLs - one each for every web service that the test should attempt to access. if each WSDL URL configured requires special permissions for logging in, then, you need to configure the test with separate credentials for logging into every WSDL URL. likewise, you need to provide instructions to the test on how to validate

Parameter	Description
	the content returned by every WSDL URL, and also set an encoding format for each wsdl url. to enable administrators to easily configure the above per WSDL URL, eg enterprise provides a special interface. to access this interface, click on the encircled '+' button alongside the url text box in the test configuration page. alternatively, you can even click on the encircled '+' button adjacent to the WSDL URL parameter in the test configuration page. to know how to use this special interface, refer to section Section **3.2.6.1**.
Operations	Once the WSDL URL(s) are specified, the operations that are offered by the web services and those that are to be monitored have to be configured. To select the required operations for monitoring, eG Enterprise provides a special interface. To access this interface, click on the encircled '+' button alongside the Operations text box in the test configuration page. Alternatively, you can even click on the encircled '+' button adjacent to the Operations parameter in the test configuration page. To know how to use this special interface, refer to section Section **3.2.6.2**.
Timeout	Specify the duration (in seconds) for which this test should wait for a response from the server. If there is no response from the server beyond the configured duration, the test will timeout. By default, this is set to 30 seconds.
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
WSDL url availability	Indicates whether the web service was able to respond successfully to the query made by the	Percent	Availability failures could be caused by several factors such as the web service process(es) being down, the web service being misconfigured, a

Measurement	Description	Measurement Unit	Interpretation
	test.		network failure, etc. Temporary unavailability may also occur if the web service is overloaded. Availability is determined based on the response code returned by the service. A response code between 200 to 300 indicates that the service is available.
WSDL response time	Indicates the time taken by the eG agent to get the configured web service.	Secs	Response time being high denotes a problem. Poor response times may be due to the service being overloaded or misconfigured. If the URL accessed involves the generation of dynamic content by the service, backend problems (e.g., an overload at the application server or a database failure) can also result in an increase in response time.
Port status	Indicates whether/not the port of the web server is reachable.		The values reported by this measure and the corresponding numeric equivalents are listed in the table below: **Measure Values** / **Numeric Values**: Yes = 1; No = 0 **Note:** By default, this measure reports the above-mentioned Measure Values to indicate whether the server has been rebooted or not. In the graph of this measure however, the Measure Values are represented using the numeric equivalents only.
TCP connection availability	Indicates whether the test managed to establish a TCP connection to the	Percent	Failure to establish a TCP connection may imply that either the web server

Measurement	Description	Measurement Unit	Interpretation
	server.		process is not up, or that the process is not operating correctly. In some cases of extreme overload, the failure to establish a TCP connection may be a transient condition. As the load subsides, the server may start functioning properly again.
TCP connect time	This measure quantifies the time for establishing a TCP connection to the web server host.	Secs	Typically, the TCP connection establishment must be very small (of the order of a few milliseconds). Since TCP connection establishment is handled at the OS-level, rather than by the application, an increase in this value signifies a system-level bottleneck on the host that supports the web server.
Server response time	Indicates the time period between when the connection was established and when the web server sent back a response header to the client.	Secs	While the total response time may depend on several factors, this measure is typically, a very good indicator of a server bottleneck (e.g., because all the available server threads or processes are in use).
Response code	The response code returned by the web server for the simulated request.	Number	A value between 200 and 300 indicates a good response. A 4xx value indicates a problem with the requested content (eg., page not found). A 5xx value indicates a server error.
Service availability	Indicates whether/not the web service is available.	Percent	A value of 100 indicates that the web service is available and a value of 0 indicates that the web service is not available.
Operation status	Indicates whether/not the configured operation is present in the web service.		This measure will not report metrics if the OPERATION parameter in the test configuration page is none in the test configuration page.
Operation Content	Indicates the response	Number	A value between 200 and 300 indicates

Measurement	Description	Measurement Unit	Interpretation
length	code returned by the server for the simulated request.		a good response. A 4xx value indicates a problem with the requested content (e.g., page not found). A 5xx value indicates a server error. This measure will not report metrics if the OPERATION parameter in the test configuration page is none or if an invalid Value is specified or if the Value is not specified in the HTML View tab while configuring the operation for monitoring in the test configuration page.
Operation Content validity	This measure validates whether the operation was successful in executing the request made to it.	Percent	A value of 100% indicates that the content returned by the test is valid. A value of 0% indicates that the content may not be valid. This capability for content validation is especially important for multi-tier web applications. For example, a user may not be able to login to the web site but the server may reply back with a valid HTML page where in the error message, say, “Invalid Login” is reported. In this case, the availability will be 100 % (since we got a valid HTML response). If the test is configured such that the content parameter should exclude the string “Invalid Login”, in the above scenario content validity would have a value 0. This measure will not report metrics if the Operation parameter in the test configuration page is none or if an invalid Value is specified or if the Value is not specified in the HTML View tab while configuring the operation for monitoring in the test configuration page.

Measurement	Description	Measurement Unit	Interpretation
Operation execution time	Indicates the time taken to invoke the configured operation in the web service.	Secs	This measure will not report metrics if the Operation parameter in the test configuration page is none or if an invalid Value is specified or if the Value is not specified in the HTML View tab while configuring the operation for monitoring in the test configuration page.

3.2.6.1 Configuring Multiple WSDL URLs for Monitoring

In order to enable the eG agent to connect to multiple WSDL URLs and pull out the required metrics from them, the eG administrative interface provides a special page using which different WSDL URLs and their corresponding operations that need to be monitored can be specified. To configure the WSDL URLs, do the following:

Figure 3.16: Configuring the WebService test

1. Click on the encircled '+' button alongside the WSDL URL text box. Figure 3.17 will then appear.

Figure 3.17: The WebService URL Configuration page

2. Specify the following in Figure 3.17:
    - **Name:** Specify a unique name by which the WSDL URL you will be specifiying shortly will be referred to across the eG user interface. This is the name that will appear as the descriptor of this test.
    - **URL:** Enter the WSDL URL of the web service that this test should access.
    - **Username** and **Password:** These parameters are to be set only if a specific user name / password has to be specified to login to the web service (i.e., WSDL URL) that you have configured for monitoring. In this case, provide valid login credentials using the **Username** and **Password** text boxes. If the server on which **WebService** test executes supports 'Anonymous user access', then these parameters will take either of the following values:

- A valid Username and Password for the configured WSDL URL
- *none* in both the **Username** and **Password** text boxes of the configured WSDL URL, if no user authorization is required
- Some servers however, support NTLM (Integrated Windows) authentication, where valid login credentials are mandatory. In other words, a *none* specification will not be supported by such servers. Therefore, in this case, against each configured WSDL URL, you will have to provide a valid **Username** in the format: *domainname\username*, followed by a valid **Password**.
- Please be sure to check if your web service requires HTTP authentication while configuring this parameter. HTTP authentication typically involves a separate pop-up window when you try to access the page. Many services use HTTP POST for obtaining the user name and password and validating the user login. In such cases, the username and password have to be provided as part of the POST information and NOT as part of the **CREDENTIALS** specification for the WebService test.

- **Content**: The **Content** parameter has to be configured with an instruction:value pair that will be used to validate the content being returned by the test. If the **Content** value is *None*, no validation is performed. On the other hand, if you pick the *Include* option from the **Content** list, it indicates to the test that for the content returned by the web server to be valid, the content must include the specified value (a simple string search is done in this case). This value should be specified in the adjacent text box. Similarly, if the *Exclude* option is chosen from the **Content** drop-down, it indicates to the test that the server's output is valid if it does not contain the value specified in the adjacent text box. The *Include* or *Exclude* value you specify in the text box can include wildcard characters. For example, an *Include* instruction can be **Home page**.
- **Encoding**: Sometimes the eG agent has to parse the **WSDL URL** content with specific encoding other than the default (ISO-8859-1) encoding. In such a case, specify the type of encoding using which the eG agent can parse the WSDL URL content in the **Encoding** text box. By default, this value is *none*.

3. Similarly, you can add multiple URL specifications by clicking the **Add More** button. To remove a WSDL URL specification, click on the encircled '-' button corresponding to it. To clear all **WSDL URL** specifications, click the **Clear** button. To update all the changes you made, click the **Update** button.
4. Once **Update** is clicked, you will return to the test configuration page as shown in Figure 3.16. The **WSDL URL** text box in the test configuration page will display just the **Name**s - i.e., the unique display names - that you may have configured for the multiple WSDL URLs, as a

comma-separated list. To view the complete WSDL URL specification, click the encircled '+' button alongside the **WSDL URL** text box, once again.

3.2.6.2 Configuring Multiple Operations for Monitoring - WebServiceTest

By default, the **WebServiceTest** test will be configured with the WSDL URLs that offer the web services that are to be monitored. To configure the operations that are offered by the WSDL URLs, do the following:

1. Click on the encircled '+' button alongside the Operations text box as shown in Figure 3.16. Figure 3.18 will then appear.

Figure 3.18: Configuring the Web Service Operation

2. Specify the following in Figure 3.18:

   - **Manager/Agent for accessing WSDL URL:** Select the eG agent or the eG Manager that is authorized to access the configured WSDL URL from this list.
   - **WSDL URL:** Once the eG agent/eG Manager is chosen from the *Manager/Agent for accessing WSDL URL* list, this list will be populated automatically with all the WSDL URLs specified in the **WSDL URL** text box (See Figure 3.16). Select the **WSDL URL** of your choice from this list.
   - **Services:** The web services offered by the chosen WSDL URL will then be populated in this list. Select a service of your choice from this list.
     - The operations that are offered by the chosen service will then be populated in the DEFINED Operations list. To monitor a chosen operation, select the operation and click the < button. This will move the chosen operation to the **MONITORED OPERATIONS** list.
     - Click the **Configure** button to save the changes.
     - The eG agent uses SOAP requests to obtain the necessary metrics from the web service. Once the operation is configured, the XML View of the SOAP Request corresponding to the chosen operation will be generated and listed in the **XML View** tab. Likewise, the **HTML View** tab lists the **SOAP Parameter** that is passed to collect the required metrics for the chosen operation.
     - To obtain operation-level statistics, it is important to specify a valid value in the **VALUE** text box of the **HTML View** tab as shown in Figure 3.19. Each time the test is executed, this value

will be provided as an input to the chosen operation.

Figure 3.19: Specifying the value for the chosen operation

- Click the **Save and Cofigure More** button to save the changes made.
- If you wish to verify if the **VALUE** specified in the **HTML View** tab is valid, then you can do so by clicking the **Send Request** button. Figure 3.19 will then appear. If the value specified in the **VALUE** text box is indeed valid, then the operation will be performed on the value and the result will be specified. For example, if your chosen operation is *FahrenheittoCelsius*, the **SOAP Parameter** is *Farenheit* and the value that you wish to convert is *100*, the result will be specified in the **WEB SERVICE RESPONSE** pop up window as below: *<FahrenheitToCelsiusResult>37.7777777777778</FahrenheitToCelsiusResult>*

Figure 3.20: The value that appears when the operation is performed successfully

- If you have specified an invalid value, then a message as follows will be displayed in the pop up window: *<FahrenheitToCelsiusResult>Error</FahrenheitToCelsiusResult>*

Figure 3.21: An Error appearing during value conversion

- If you do not specify a **VALUE** or specify an invalid value, operation-level statistics will not be collected by the eG agent and such metrics will not be available in the eG monitoring interface.

3. Similarly, you can configure multiple Operations by clicking the **Configure** button in Figure 3.18. To remove an operation, select the operation from the **MONITORED OPERATION** list and click the **>** button.

Once **Save and Configure More** button is clicked, you will return to the test configuration page (see Figure 3.18). The Operations text box in the test configuration page will display just the operations, as a comma-separated list. To view the complete operation specification, click the encircled '+' button alongside the Operations text box, once again.

3.3 Troubleshooting

If the measures for a JRun server being monitored do not show up, then check whether the HomeDir parameter is set in the proper format. For example, if the JRun server is installed at D:\program files\allaire\jrun, HomeDir has to be set to D:\progra~1\allaire\jrun.

Sometimes measures might remain unchanged for a considerable period of time. In such cases, check the following:

- Check whether the metrics logging is enabled
- Verify whether the server was restarted after enabling

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.