

Diagnostics FAQ

December 2013

1. Since the Diagnostics 9.x .NET Agent doesn't support .NET 1.1 how can I monitor .NET 1.1 applications?

Use the Diagnostics 8.0x .NET Agent for applications running under .NET 1.1. You can have an 8.0x .NET probe running and communicating to a Diagnostics 9.x Server.

2. If I have both .NET 1.1 and .NET 2.0 on the same system can I have multiple .NET agent versions on the same machine?

You cannot have more than 1 version of the .NET agent on a single machine. The 9.x .NET agent has new profiler interfaces and support for .NET 1.1 had to be dropped in order to implement these new features using the .NET 2.0 framework libraries.

3. I don't see my .NET probe in the Diagnostics commander's System Health registrar and I tried to enable the probe and restart IIS but still don't see it in system health graph.

The .NET probe is only active and will only show up in the system health graph when the application it is monitoring is running. Windows shuts the probe down when the application is not being used, even though IIS is running.

4. Since the Diagnostics 9.23 Java Agent doesn't support Java 1.4, how can I monitor my legacy applications? You can have a 9.20 or 9.21 Java probe running and communicating to a Diagnostics 9.23 Server.

5. Why is the probe overhead so high on IBM JVMs?

The IBM (J0) JVM 1.5.0 has a defect that causes JVM performance degradation. The recommended way to use Diagnostics with this JVM is to use -Xbootclasspath option instead of -javaagent option.

6. Why can't I do dynamic instrumentation of IBM JVMs?

The IBM (J0) JVM 1.5.0 has a defect that causes JVM performance degradation. The recommended way to use Diagnostics with this JVM is to use -Xbootclasspath option instead of -javaagent option.

Unfortunately, dynamic instrumentation requires -javaagent. In development environment, you may still want to use this option, if the high overhead can be tolerated. Note that the overhead is high all the time; it has nothing to do with Diagnostics activities.

If you are in a development environment and want to use the -javaagent option there is another limitation of the IBM J9 JVM where "-javaagent" and "-agentpath" are mutually exclusive - only one of them will take effect. So, by using -javaagent you will gain dynamic instrumentation capability,

but lose HeapWalker feature and good performance. The safest way to invoke Diagnostics in this environment would be simply this:

```
-javaagent:/opt/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/lib/probeagent.jar  
-Xshareclasses:none -Dprobe.id=gblabl14
```

If you need heap diagnostics, you can try to use HeapBreakdown. This would require setting up LD_LIBRARY_PATH to include /opt/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/lib/x86-linux and then using this invocation:

```
-javaagent:/opt/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/lib/probeagent.jar  
-Xrunheapdump -Xshareclasses:none -Dprobe.id=gblabl14
```

7. What if there are different JVMs how do I know which to pick to instrument in the JRE instrumenter?

Use JREInstrumenter in Automatic Explicit or Implicit mode. If you cannot do that, read on. Many application servers use launcher utilities that can and often do use a different JVMs to start the actual, final JVM (which leads to MUCH confusion as to what JVM to instrument, what versions of Java are running, etc.)

If this is a Unix - linux system where you can see the command line the JVM is using then you can do a `ps -ef | grep java` and you should be able to see the application server's entire command line.

For example, when entering this command on my test machine where I have more than one java process running, the 1st one, because it does not specify a path is using the default JRE installed on the machine:

```
sdd 5597 5576 0 11:50 ? 00:00:09 java
```

One of the other processes on my test box DOES specify a path and therefore it is using a DIFFERENT run time:

```
sdd 24986 24983 0 Nov28 ? 00:02:34 /usr/bin/java
```

You can check the java version by copying the path and java command and add `-version` to the end of it. For that second process this command will display the JRE version:

```
/usr/bin/java -version
```

8. How does Diagnostics support clustered JVMs?

To see performance for the cluster, first put all probes for the JVMs in the cluster into the same probe group. Then you can use the Aggregate Server Request view to see performance of the whole cluster and drill down to individual probes.

9. Can Diagnostics help identify load balancing issues in a cluster?

Assuming you've put all probes for the JVMs in the cluster into the same probe group then in the Aggregate Server Request view, you can add the count metric to the entity table which tells you the total number of requests across the cluster. You can drill into the aggregate server request to see the server request performance in each JVM. Again use the count metric to see the number of server request instances for each JVM.

10. What should I consider when choosing an approach for monitoring multiple JVMs on a single server? There are two options: 1) installing one agent per server and then configuring multiple probes for the JVMs 2) installing an agent for each JVM.

If you want to minimize the number of agents you have to install and setup you can plan to install just one agent on the server. This works best if the JVMs on the server are similar enough that you can use the same agent configuration files and libraries etc. for all the JVMs. In this scenario there is only one probe.properties file (for the agent) so you use command line overrides of the values in this file along with running the JRE instrumenter for each JVM to configure unique probe names and instrumentation for each JVM.

Another approach if you have very different JVMs on the same system or you have very different agent configuration requirements is to install the agent multiple times and define an instance of a probe for each agent by setting the probe's id property in the probe.properties file for each agent.

Refer to the Java Agent Guide section on "Configure Monitoring of Multiple Java Processes on an Application Server".

11. If you are setting up multiple probes on the same system do you have to give them unique names, and if so, how do you do this? How do you set up unique probe names?

Diagnostics relies on unique probe names. Some functionality will not work if they are not unique.

When you install an agent a default probe name is assigned. You can change the name by modifying the id property in the <agent install dir>/etc/probe.properties file. If you have multiple JVMs on a system and have installed the agent multiple times you can set the id property in the probe.properties file in each agent's /etc directory.

Id=<uniqueProbeName>

Note: This approach will not work if you have configured multiple probes/JVMs based on a single agent installation. Because there is only one probe.properties file, you can't assign more than one id=property.

What you can do in this scenario is override the probe name in probe.properties by specifying a unique probe name for each of the JVMs probes. This is done using the Java command line or startup script (in the Generic JVM arguments section).

-Dprobe.id=<Unique_Probe_Name>

You can also use the %0 option to generate a unique probe id rather than assigning a specific name. For example you might want to use this approach in a clustered environment where a single startup script is used to start multiple application server instances.

-Dprobe.id=<probeName>%0

On Windows, use %%0. Use the first % to escape the second %. The %0 is replaced dynamically with a number to create a unique probe name for each probe; for example, <probeName>0, <probeName>1, and so on.

Refer to the Java Agent Guide section on “Configure Monitoring of Multiple Java Processes on an Application Server”.

12. I see my probe in the Enterprise UI but my Server Request and SQL views are empty. Why don't I see my servlet or message bean or EJB...?

First check the appropriate Java Agent monitoring profile in etc/defaults to see if they have been configured for the profile you are using.

If the items still do not appear, in dispatcher.properties check the following properties:

- minimum.fragment.latency = 51ms (default) Tune this value
- minimum.sql.latency = 1s (default) Tune this value

13. What is the difference between Total CPU Utilization and Normalized CPU Utilization?

Normalized CPU Utilization on the probe is CPU utilization for the application server process divided by the number of logical processors/cores.

Total CPU Utilization is on all cores added up so it can be greater than 100.

So for example if Total CPU Utilization is 4% and you have 4 CPUs/Cores, the Normalized CPU Utilization is 1%.

Both metrics represent the same thing – the CPU consumption by the probe process. This includes the JVM, the Java application, the probe, and all the native libraries which may be used by the JVM or the application. This is the sum of CPU utilization by all threads belonging to the process.

If the system has N CPU/Cores then

Normalized CPU Utilization = Total CPU Utilization/N

On a single CPU system the metrics have identical values.

14. Where can I find the Diagnostics support matrix?

Customer visible path is Diagnostics Product Availability Matrix at http://support.openview.hp.com/sc/support_matrices.jsp

15. I've installed a probe/collector and configured its mediator server but I don't see the probe in the Diagnostics Enterprise UI. What should I do?

- Check the collector log. Is the collector connecting to the database or whatever it is collecting from?
- Did you give the probe/collector a unique name? if you are working with two probes at the same box same probe installation, make sure you use the jvm options `-Dprobe.id=xxx` to differentiate them.
- Do you see probe.log under the configure probe name in the log directory? If not, you have not specified the necessary Java command line options to load the Agent.
- Check that the mediator process is running and available (communication and firewall) from the probe machine
- Check registrar - system health view
- Check commander's server.log

16. What does the Average Downtime metric mean? If an app server (hence the probe) is down for more than 2 days but downtime metric when viewing data for the last 3 months is 26m, this seems incorrect.

The Average downtime is a metric that measures how quickly the probe is started up after a failure (crash) and it is simply the arithmetic average of all downtime period lengths over the viewing data periods. This metric is useful only for viewing historical data.

This metric is collected upon probe restart or upon probe reconnecting to the mediator so if the probe is currently down the metric does not report that until the probe restarts.

So for example if the probe has been down for the last 2 days and didn't restart, these 2 days are not taken into account when calculating Average Downtime since this is how long it takes to restart not how long the probe was down.

If you want to check the time when a probe becomes unavailable, for example if the probe is down for maintenance, you may consider probe availability metrics.

If you want to scrape the logs on the mediators, there is a message logged each time the probe connects:

```
2010-11-07 14:55:43,929: INFO data_in : Successfully pulled (ProbeTrendsPullerTask) data from ProbeTrendsPullerTask
```

and if the probe goes offline and the mediator cannot access it, the log shows:

```
2010-11-07 14:56:30,080: INFO data_in : Removing ProbePullerTask ProbeTrendsPullerTask
```

17. I am concerned about the overhead of instrumentation. What mechanisms does Diagnostics provide to reduce instrumentation overhead?

First of all make sure your instrumenting is appropriate – for example the basic recommendation is not to instrument `get/set` calls. These are simply returning or setting a single value, very fast. For transactions with a very small transaction time you wouldn't need to instrument for performance.

Then note that Diagnostics is designed to use the level of instrumentation that will provide adequate information to troubleshoot a temporary or hard to reproduce performance issue while imposing a low overhead that can be tolerated in most production environments.

To achieve this goal, Diagnostics provides two mechanisms which automatically adjust data collection in response to the performance characteristics of the currently executing server request.

The first such mechanism is latency-based trimming. If a particular invocation of an instrumented method is fast, the invocation is not reported (there will be no corresponding node in the Call Profile). This cuts the overhead substantially, as the Diagnostics Agent does not have to create the necessary object and place it in the call tree. At the same time, it is assumed that such fast calls are of no interest to the user who is interested in pinpointing performance issues. You can adjust the reporting threshold (51 ms by default) to eliminate some of these types of fast calls (presented by very thin bars in the call profile). These calls have relatively high overhead, and probably do not provide any useful information which can help diagnose performance issues.

Another automatic data collection mechanism is stack trace sampling. This feature reports long running methods even if they are not instrumented. Thus by enabling this feature, and tuning it to provide adequate level of information, the user can turn off some of the instrumentation and trust that any potential performance issues in this module will be reported by stack trace sampling.

As far as light-weight code injection, we do exactly that. Our instrumentation is as light-weight as possible. One should realize though that a major portion of the overhead is caused just by taking a timestamp (which is necessary to calculate the latency). Also see the section on Monitoring Profiles in the Java Agent Guide.

18. What are the metrics Diagnostics collects out of the box?

The document of what metrics are collected is the `metrics.config` file for Java agent and `metrics.config` file for .Net agent.

What metrics Diagnostics collects is dependent in part on what application server you are running (WAS, WebLogic, Jboss, IIS...).

If looking at WAS the metrics Diagnostics collects are dependent on the PMI setting of WAS. Diagnostics Java Agent can collect any JMX and PMI metrics.

The Java Agent metrics.config file has a feature to write a list of all the available metrics for each JMX collector into a file. When the **default.dump.available.metrics** property in the metrics.config file is set to true, the probe will write this list of available metrics to text files in the probe log directory. The files are named as follows: <probe_install_dir>/log/<probe-id>/jmx_metrics_<collector-name>.txt. Interrogate all the metrics available to the probe inside the JVM. The output can be edited into new metrics.config entries on the fly.

You can customize the metrics collection in the agent's metrics.config file. Refer to the Agent Guide chapters on "Configuring Diagnostics Metrics Collectors".

A system metrics collector is installed with the Java Agent and the .NET Agent. It gathers system level metrics such as CPU usage and memory usage from the agent's host. You can customize the system metrics collection in the metrics.config file.

19. What should I do with the probe group?

- Probe group is used to aggregate data across clusters.
- Aggregate Server Request view is all requests across cluster
- SQL statement view is all SQL across cluster

20. How can I see how many users are accessing the Diagnostics Enterprise UI?

You can get a list of active users seen by the Diagnostics server in the last 60 seconds. And you can see the Queries/sec indicating how much load the user generates with summary or trend queries.

From the main Diagnostics UI select **Configure Diagnostics** and the Components page is displayed. (You can also access this Components page by selecting the Maintenance link in any Diagnostics view). Select the **query** link and then select the **Active Users** link at the bottom of that page to display a list of active users. Also this data is under Mercury System groupby.

21. Why am I not seeing information for the method I instrumented via auto_detect.points?

You can check **detailReport.txt** in the log directory of the probe and make sure the method you are instrumenting uses the intended instrumentation point. If multiple instrumentation points satisfy the conditions for a particular method, the probe either uses the point with higher **priority** or else can't predict which point will be used.

22. Is there a "simple" way to trim the unique server requests to reduce impact of probe on server processing load and also on server disk space?

An effective way will be using the `uri.pattern.replace` property in `<probe_install_dir>/etc/dynamic.properties` file. Refer to the comments in the properties file. This works only for `http/https` URIs. It uses regular expressions based on Perl syntax.

23. I don't see any system metrics (Disk bytes/s, Network bytes/s, Page In/s and Memory percentage) even though I have the SAP Collector on that system?

The SAP collector does not report system metrics. To get them, install a Java agent on the desired host and run the `bin/startAgent.sh` to collect these system level metrics.

24. Explanation of Diagnostics thresholds?

Each of the numeric metric data for an entity (CPU of a host, heap used in a VM...) can have a threshold value set. Threshold is evaluated against the metric data points received, usually every 5 seconds. The metric with a threshold set will have one of the following status levels: Green, Yellow and Red. The entity's status is derived from all its metric statuses according to worst-child rules (if any metric for the entity is red, the entity is red).

As long as the metric value does not exceed the threshold the status remains Green. If 3 or more metric data points are beyond the threshold the status turns to Yellow. If the average metric value within the last 5 minutes is beyond the threshold the status becomes Red. Once the 5 minute average goes below the threshold the status becomes Green again. Note that Diagnostics status does not revert to Yellow it goes directly back to Green.

The threshold values for metrics are configurable in the UI (details pane) and some metrics also have default thresholds set. The default threshold configuration is set in the server's `etc` directory in `thresholds.configuration`.

If you need to set thresholds on specific methods you would want to add a separate entry in the `points` file for each method and this will allow you to set up thresholds and alerts for the specific method.

25. Questions on Diagnostics and SiteScope integration.

What port does the Diagnostics/SiteScope integration use? You point SiteScope to a Diagnostics MEDIATOR on the standard 2006 port. For example you'd set Receiver URL to:

<http://meditor.customer.com:2006/metricdata/siteScopeData>.

And once I tag an existing SiteScope monitor with this Diagnostics integration will I need to do any restarts or touch my files? No there is no need to bounce any servers or touch any files.

When you first try to view SiteScope data in the Diagnostics External Monitors view, by default the monitor's status is gray and no data is graphed. To see a status (red, yellow, green), you must first

set a threshold on a metric (in the details pane). To see data in the graph, you must first select a metric to be charted (in the details pane).

26. More information on the threads metrics.

I'm running an application where I see a lot of threads in a blocked state and I'd like to know the exact amount of time each thread keeps blocked. What I'm interested in knowing is total execution time of a server request and time spent in blocked state for that server request.

The thread metrics shown in the Profiler's Threads tab are collected independently of the server requests.

The values shown in the table are cumulative metrics for:

- CPU time spent in OS kernel
- CPU time spent in user mode
- Time spent in waiting state (in `Object.wait(...)`)
- Time spent in blocked mode (lock contention for synchronized methods or blocks)

The values can only increase and they correspond to the usage since the thread creation.

The graph shows the difference between the values of these metrics between the last two thread snapshots. Graph makes little sense unless you enable automatic update with a constant frequency.

When a thread is returned to a pool its ID doesn't change.

Blocked time is also shown for Call Profiles, as one of the latency metrics for the topmost call profile entry.

27. What is the overhead of the Diagnostics Oracle collector? Does the oracle collector actively query the Oracle SID/Database or does it passively collect metrics?

The overhead should be negligible. The Oracle collector runs a couple of SQL queries from Java periodically (the interval is configurable) to collect the performance metrics. Also the collector can be installed on any machine so it does not have to be on the Oracle host.

28. What Java Agent changes can be made without requiring a restart?

In general all of the setting in the `dynamic.properties` files can be changed without restarting. There are a few other settings you can change without restarting and this will be noted in a property file's comments.

29. Is it possible to configure Diagnostics to send immediate alerts instead of waiting for the 5 minute threshold to be crossed?

No this is not possible with Diagnostics.

30. How can I monitor if my applications are producing dynamic server requests which can cause symbol explosion in Diagnostics?

Symbol explosion from dynamic or ever changing server requests can cause performance problems in the Diagnostics Server, and agents should be configured in a way to prevent this using URI collapsing, filtering and symbol name replacement. The Diagnostics Server has the ability to detect if there is an unusual growth in the number of server requests being created. Often, this can clue the user into problematic dynamic server request names or URIs which should be addressed.

There are several configurations in server.properties starting with "watchdog.server_request_rate" for setting conditions of when a warning email and log message are produced.

If and when such a warning is produced, it will indicate the number of server requests created in the past 1 day, 7 days, and 1 month. This will show how there appears to be an unusual growth on the number of server requests being created when they typically should become fairly static over time. The server requests should be examined to see if there appears to be anything dynamic about them, and it should be considered to make changes in the agents to do symbol name replacement to fold in the dynamic parts of the server requests.

31. I know there are symbol folding and trimming options on the agents, but can any kind of symbol folding or trimming be done at the mediator?

In Diagnostics 9.23, at the mediator, you can now do similar symbol pattern replacement and folding as seen in the agents. Keeping in mind that it is still more efficient to do this kind of replacement and folding earlier on in the agents, having the same ability in the mediator allows for both a more centralized way of handling symbol explosion and a way to take care of trimming symbols from older agents that may not have the trimming feature. These trimming/folding settings can be found in server.properties under the "Probe data trimming / folding" section.