Resin Cookbook - Article: **Resin Debugging**

The Resin Application Server health system provides many useful tools to monitor, report, and alert on the health of your application server. Monitoring of all the typical metrics such as high cpu, low memory, deadlocked threads, etc, is pre-configured for you in health.xml.

We also include appropriately conservative remediation actions in health.xml, such as triggering thread dumps, heap dumps, and restarts when necessary. It's up to you to tweak these settings to increase or decrease the aggressiveness of the health system as you see appropriate.

Any numeric attribute of any MBean in JMX can be configured as Meter in Resin, which then enables:

Persistent historical tracking
Visual graphing in resin-admin
Visual graphing in PDF reports
Cluster wide reporting
Health monitoring
Anomaly analysis and logging
Triggering health actions (heap dump, thread dump, restart, etc)

Sometimes it is necessary to dig deeper when analyzing what is going wrong in a Resin deployment. In these cases the following three areas are most helpful for resolving a possible issue:

1. Resin log files
2. Resin PDF Report
3. Extending health.xml with custom actions for tracking anomalies

1. Resin Log File

Resin's main server log is the primary source of information for a deployment. The default ivm-server log path can be set by adding a path attribute to the <log-handler> in resin.xml as such:

```
<log-handler name="" level="all" path="data/log/resin/jvm-${serverId}.log"
        timestamp="[%y-%m-%d %H:%M:%S.%s]"
        format=" {${thread}} ${log.message}" rollover-period="30D"/>
```

By default, the resin.properties defines the log level to "fine". While this is useful for most cases, during a debugging session it is useful to set the logging level to a greater granularity which can be done in two ways:

1. Altering the resin.properties **log_level** line
2. Using the command line to temporarily set a new logging level as such:
   2.1. Change the logging level temporarily with log-level. The java.util.logging level will change to the new value:

Example: setting log level
unix> resinctl log-level --finer --active-time 5s com.mycom.mypkg

Log level is set to 'FINER', active time 5 seconds: {root}, com.caucho
log-level options

Log levels are defined as follows:

off		turn off logging
severe	log.severe("...")	a major failure which prevents normal program execution, e.g. a web-app failing to start or a server restart
warning	log.warning("...")	a serious issue, likely causing incorrect behavior, like a 500 response code to a browser
info	log.info("...")	major lifecycle events, like a web-app starting
config	log.config("...")	detailed configuration logging
fine	log.fine("...")	debugging at a user level, i.e. for someone not familiar with the source code being debugged
finer	log.finer("...")	detailed debugging for a developer of the code being debugged
finest	log.finest("...")	events not normally debugged, e.g. expected exceptions logged to avoid completely swallowing, or Hessian or XML protocol parsing
all		all messages should be logged

If Resin restarted, you should see one of the following Exit Codes in the log file. Descriptions are provided alongside in the following table:

```
{
  OK,        // normal exit
  EXIT_1,
  FAIL_SAFE_HALT, // fail safe shutdown exit (normal shutdown frozen)
  BAD_CONFIG, // config error forces restart
  BIND, // failed to bind to TCP port
  MODIFIED, // configuration has changed
  MEMORY, // out of memory
  THREAD, // thread failure (generally failure to create new thread)
  ALARM_FREEZE, // the alarm thread has frozen
  HEALTH, // health check failed
  NETWORK, // BAM network failure
  WATCHDOG_EXIT,  // watchdog requested exit (the "stop" command)
  CPU, // CPU overuse
  UNKNOWN,
  UNKNOWN_ARGUMENT,
}
```

A useful technique is to enable full debug logging to track down a problem:

```
 <log-handler name="" level="finer" path="log/debug.log"
      timestamp="[%H:%M:%S.%s]"
      rollover-period="1h" rollover-count="1"/>
```

If the user is unfamiliar with other logged exceptions, the can be emailed to resin-user@caucho.com for users with a support plan or presales@caucho.com for users no support plan.

2. Resin PDF Report

A Resin PDF report is a nicely packaged overview of the system, where users can quickly get a sense of what is going on within the deployment.

Resin can generate two slightly different PDF health reports: Snapshot and Watchdog reports. A Snapshot Report captures a “snapshot” of Resin at the current point in time. A Watchdog Report aggregates as much information as available about Resin at a previous point in time. Watchdog reports can also be thought of as a “post-mortem” or “restart” report, as they usually are generated immediately after an unexpected server restart.

Gathered statistics:

Report times - shows the time the report was generated and the time range of the report
Environment - shows the JVM, operating system, and CPU information
System Resources - shows JVM heap, physical memory, swap and file descriptor summary
Resin Instance - shows the directories and files and Resin version. Also shows any startup messages.
Licenses - shows the current Resin Pro licenses
TCP Ports - status of the TCP listeners and threads at the time the snapshot was taken
Health - The HealthCheck status of the server and a graph showing the status over time.
Recent Warnings - Shows log warning messages before the snapshot was taken
Recent Anomalies - Shows any anomalous behavior (like thread spikes) detected by Resin
Cluster Status - shows the status of all the servers in the cluster as a set of graphs
Server Graphs - graphs of the most important meters for the system over the measured time.
Heap Dump - shows the most heavy memory usage for the system. You can use this as an initial
memory debugging.

CPU profile - optional, when a profile is taken, will show the most heavily used methods
Thread dump - shows the state of all threads in the system when the dump was taken
Full log - shows all of the java.util log warnings over the time period
JMX dump - shows the full state of JMX

You can generate a PDF report in the following ways:

1. Via the command line:
   1.1. resinctl pdf-report -local -local-dir /tmp
2. Via REST
   2.1. curl 'http://localhost:8080/resin-rest/pdf-report?snapshot=true&load-pdf=true' > snapshot.pdf
3. Via the /resin-admin app
   3.1. http://localhost:8080/resin-admin/index.php

Watchdog report are usually generated automatically after any unexpected server restart. Check your logs directory for Watchdog-*.pdf files. If needed, you can generate a watchdog report at from the /resin-admin watchdog page

3. Expanding upon health.xml

Resin defines numerous actions and health checks within the health.xml file. Among them, is the extremely useful automatic analysis tool called AnomalyAnalyzer. AnomalyAnalyzer looks at the current meter value, checking for deviations from the average value. So unusual changes like a spike in blocked threads can be detected. Meters are defined within health.xml.

```
<health:AnomalyAnalyzer>
  <meter>JVM|Thread|JVM Blocked Count</meter>
```

```
  <health-event>caucho.thread.anomaly.jvm-blocked</health-event>
</health:AnomalyAnalyzer>
```

In this example we've created an AnomalyAnalyzer on the blocked thread meter we created above, and assigned it to the health event "caucho.thread.anomaly.jvm-blocked". The health-event attribute is optional. **Without a health-event, an anomaly analyzer alone will only log anomalies it detects to the resin log at WARNING level**. These alerts also show up in PDF reports. An example anomaly log is shown below:

2012-01-20 16:10:00 AnomalyAnalyzer JVMlThreadlJVM Runnable Count WARNING value=3.000, deviation=9.487 sigma mean=2.011 std=0.104 n=92.0

Resin's health system provides a set of remediation actions that you can configure to automatically execute in reaction to an anomaly. The <health-event> attribute we configured above allows us to tie health actions to a detected anomaly, as shown below:

```
<health:DumpThreads>
  <health:IfHealthEvent regexp="caucho.thread"/>
  <health:IfNotRecent time="15m"/>
</health:DumpThreads>
```

In this example we've created a DumpThreads action with 2 conditions.

The first condition, IfHealthEvent, tells the action to execute only if the health event starts with "caucho.thread".

The send condition, IfNotRecent, prevents the action from executing more than once every 15 minutes.

Greater granularity with the CPU Profiler

In some cases, it may be useful to reach the highest level of granularity by having a health action enable the Resin CPU profiler to see exactly which threads are taking the most CPU processing time:

Resin's profiler should be able to run in a lightweight mode by changing the "sampling-rate" of the profiler. You can use either the <health:IfCron> or <health:IfNotRecent> tags to specify the exact time(s) you want the profiler to run. By doing this, the profiler will run continuously without introducing too much overhead to the stack.

You can define an ActionSequence in your health.xml file similar to the following (Uncomment either the IfCron or IfNotRecent):

```
<health:ActionSequence>
   <health:StartProfiler sampling-rate="100ms" wait="false"/>
   <health:DumpThreads/>
   <health:ScoreboardReport/>
   <!-- <health:IfCron value="* * * * *"/> -->
   <!-- <health:IfNotRecent time="10m"/> -->
  </health:ActionSequence>
```

Take a look at http://www.caucho.com/resin-4.0/admin/health-checking.xtp#Healthactions to see the available actions. It's recommended to avoid dumping the JMX and the heap as they are expensive calls and the heap dump will temporarily freeze the JVM.