Resource Monitor User Guide revision 0.91

HOW-TO Implement
A Resource Monitor Application
for LINUX

(using shell scripts, C or C++)

Author: Ken Banks (ken.banks@intel.com)

Key Contributor: Zhu, Yi (yi.zhu@intel.com)

Abstract

This paper describes the interfaces for the Resource Monitor and provides examples of their usage.

Disclaimers

THE INFORMATION IS FURNISHED FOR INFORMATIONAL USE ONLY, IS SUBJECT TO CHANGE WITHOUT NOTICE, AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY INTEL CORPORATION. INTEL CORPORATION ASSUMES NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR INACCURACIES THAT MAY APPEAR IN THIS DOCUMENT OR ANY SOFTWARE THAT MAY BE PROVIDED IN ASSOCIATION WITH THIS DOCUMENT. THIS INFORMATION IS PROVIDED "AS IS" AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THE USE OF THIS INFORMATION INCLUDING WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, COMPLIANCE WITH A SPECIFICATION OR STANDARD, MERCHANTABILITY OR NONINFRINGEMENT.

Legal Notices

Contents

Introduction

The Resource Monitor (RM) is a daemon (background) process for LINUX capable of monitoring anything that can be measured in a system. System measures, or statistics, are accessed via loadable subsystem libraries. Subsystems currently included with RM provide statistics for SCSI, Ethernet (e100), kernel, file system and processes. Applications can maintain a continual state of awareness and respond to situations that may require attention by monitoring statistics on all of the available subsystem resources. RM-aware applications can empower your LINUX system with unlimited capabilities, such as...

- autonomous performance tuning
- autonomous network management
- predict, detect or prevent undesirable system states
- collect historical data for benchmarks, research or graphical views
- manage critical resource availability for high availability (HA)

This document illustrates how to implement a basic RM application using a choice of languages including C, C++ and even command shell (using XML) for administrators who require the speed of scripting over developing with a compiler. For a quick start, see the Appendices for some usable sample code.

The following Overview offers a brief orientation on the RM framework. The remainder of the Resource Monitor User Guide (RUG) offers a firm foothold for any user of RM. Using this guide, you should be able to create or instrument an existing RM-aware application with some pre-requisite knowledge of one of the three languages listed above.

Overview

RM 1.5 allows LINUX administrators to monitor system resources and generate events based on resource availability or statistical behaviors. The POSIX Event Log (PEL) provides the ability to act on these events. For instance, "percent free space" (a statistic for a disk partition) may drop below 5% generating a system event that would trigger the PEL action daemon (evlactiond) to invoke a script to free more space by purging all temporary files. This section describes how RM and PEL are used together to accomplish this or similar scenarios.

RM Entities and Types

Resource Monitor entities consist of *subsystems* that provide *statistics* for each of its available *resources* (such as disk drives or Ethernet adapters, etc) and, of course, *monitors* (for watching statistic behavior). Subsystem libraries are loaded by the RM daemon (resourcemonitord) at boot time from the paths specified in **/opt/resourcemon/rmtab**.

RM version 1.5 provides two (2) types of statistics:

1. a **counter** which can only increase in value over time and
2. a **gauge** whose value can fluctuate up and down.

Monitors can be created for any statistic made available by a subsystem for its resource(s). RM version 1.5 provides three (3) types of monitor:

1. **watermark** - records highest and lowest values, but does NOT generate any events.
2. **threshold** - generates a system event whenever a threshold exception occurs.
3. **leakybucket** - generates a system event whenever a rate-of-change is exceeded.

How It All Works (with the POSIX Event Log)

There are four unique actions that illustrate how the event log is used together with RM applications and scripts. When a monitor is created, the **resourcemonitord** assigns a unique **Monitor_ID** to be used for accessing and controlling the monitor. Applications must create/register a PEL query for the Monitor_ID in order to receive notification of threshold or leakybucket exceptions that would be detected by the monitor. When a statistic value meets a threshold condition or a leakybucket rate criteria the resourcemonitord (for *daemon* monitors) or the subsystem resource (for *inline* monitors) generates a PEL record into the POSIX system event log following the format of an RM event schema described in **MonitoringEventSchema.h**. When the Monitor_ID of this new event record matches the criteria of the query (created in step 2), PEL invokes either the response shell script or the C/C++ callback/listener function.

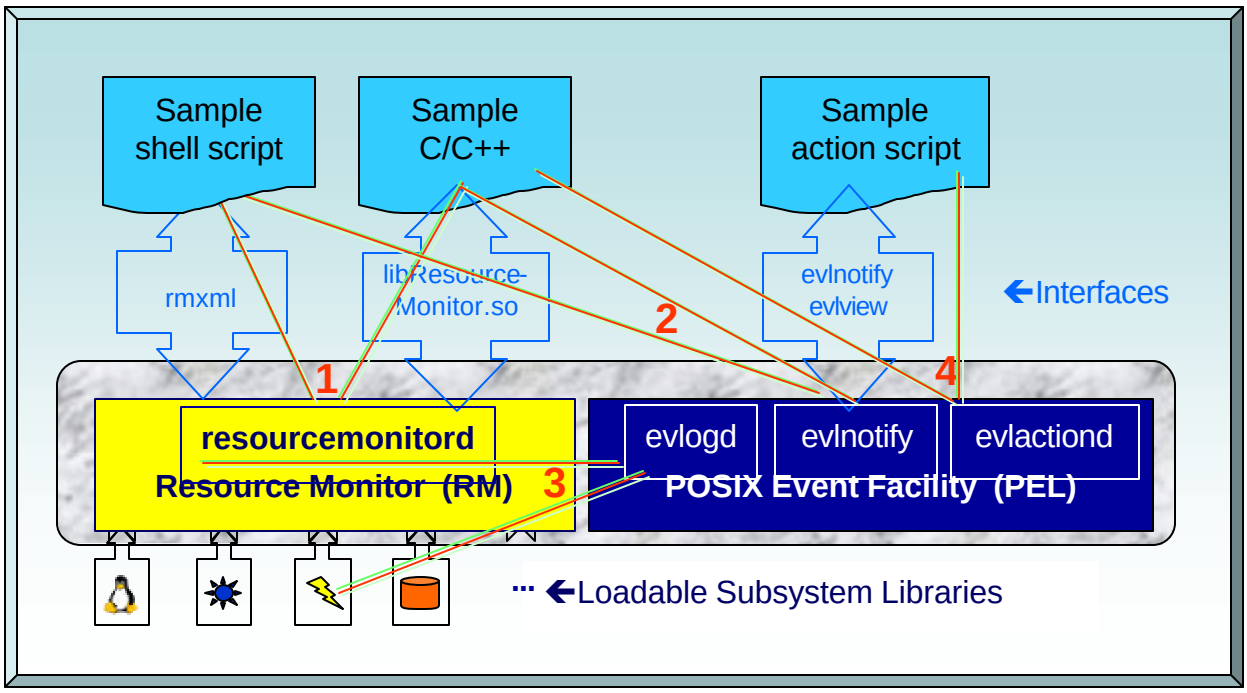

Figure 1 Steps of a Resource Monitor working scenario:

1) **Monitor creation. (Monitor_ID assigned)**
2) **Query creation. (Filter for Monitor_ID and assign listener/response)**
3) **Exception event occurs. (PEL query detects Monitor_ID of new record)**
4) **Event notification and response. (Query invokes listener function and/or script)**

NOTE: Watermark monitors do NOT generate PEL records. They must be sampled using the '**rmxml status'** command or **rmGetCurrentValue()** and **RMmonitor::GetMonitorInfo()** API's.

RESOURCE MONITOR USER GUIDE

Resource Discovery

Before creating any monitors, it may be useful to discover what is available to be monitored in the first place.

The "**rmxml discover"** command will generate XML describing all subsystems, resources and statistics. Monitors themselves are discovered using the "**rmxml state -a**" shell command. These commands are described in further detail in the Appendix.

Subsystems, resources, statistics and monitors can all be discovered using C or C++ API's provided by the RM library **(/usr/lib/libResourceMonitor.so)** as listed in the appendices.

All uniquely-identifiable entities in the Resource Monitor implementation are identified using universally-unique identifiers, or UUIDs. Subsystem UUID's and statistic id's can be very cryptic and meaningless when referenced or read literally. For example, the UUID for the kernel subsystem is **`f05d2371-d0d5-4cf9-866e-0c8e92afde66`**. Header files provide convenient compiler substitution strings for UUID values (for example **`RM_KERNEL_UUID_STRING`**). More substitution strings can be found in the header files provided for each subsystem at **/usr/include/ResourceMonitor** or location in a provided by the subsystem library developer.

Monitor Creation

Before creating a monitor it is important to select the appropriate monitor type that will accomplish your intended purpose for monitoring a statistic. It is easy to assign monitors to statistics; even those that make no sense. For example, a *low* watermark monitor (without a transform, see below) assigned to a counter statistic is not very useful since a counter will only continue to increase in value. Make sure that the type of monitor assigned and the attributes that you set are logical. The RM daemon does not second-guess any monitor attributes.

All monitors are defined by a combination of two structures:

- **rmMonitorControl** - primarily "controls" the location and logging of a monitor.
- **rmMonitorConfiguration –** configures how a specific statistic is monitored.

Minor differences exist between inline and daemon-side monitors. These are covered in descriptions of specific functions below

NOTE: Most monitor configuration parameters can't be changed once the monitor has been created. The **typeConfiguration.threshold** parameters can be changed except for its type field. All other parameter changes are invalid.

Statistic Transformations

Sometimes it is not wise to create a monitor for a raw statistic value. For example, a threshold monitor created to invoke an emergency response script whenever allocated virtual memory exceeds 507 megabytes on servers with 512 megabytes (or approximately 99% allocation) could become a nightmare, if these servers were later upgraded to 1 gigabyte. This monitor's emergency response script would then be invoked after exceeding only 50% memory allocation. Statistic transformations can prevent such problems.

A statistic transformation is an algorithm that can be applied to a raw statistic value prior to being evaluated by a watermark or threshold monitor. Allowable transformations include:

- **rmNone** - monitors the raw statistic value
- **rmChange** - monitors the change from one statistic sample to the next
- **rmPercent** - monitors the percentage of a gauge statistic
- **rmPercentChange** - montors the percent change from one sample to the next

A threshold monitor using the **rmPercent** transformation and a threshold value of 99 would always be valid for the scenario described above regardless of future memory upgrades.

Watermark Monitors

A watermark is used to record the highest and/or lowest value sampled on a raw statistic or its transformation. There are three types of rmWatermark…

1. rmHighWatermark
2. rmLowWatermark
3. rmDualWatermark

A watermark does NOT generate a system event. To discover current watermark values recorded by monitors that you have created, you must use the **rmxml state** command or a library API (**`rmGetMonitorState`** or **`RMmonitor::rmGetMonitorState`**).

Watermark monitors with **rmChange** and **rmPercentChange** transforms can be helpful when trying to derive good attributes for both Threshold and LeakyBucket monitors.

NOTE: rmSetMonitorConfiguration() has no effect for a watermark monitor.

Threshold Monitors

A threshold is used to detect and respond to statistical conditions about a specific subsystem resource. There are two types of rmThreshold…

1. rmThreshold
2. rmBidirectionalThreshold

A threshold monitor generates an event for every sample that meets its test condition. For daemon monitors, eventing can be throttled with the parameters found in **rmMonitorControl**. If the **loggingRate** attribute is set to 0, then only one event is generated when the test condition is first met changing the monitor's state to **pauseNotification**. .

A bi-directional threshold will generate one additional cancel event after its test condition is no longer met.

NOTE: All threshold configuration parameters except the type can be modified after a daemon monitor is created.

Leakybucket Monitors

A "leaky bucket" is typically used to respond before you have run out of a specific resource. For example,

If the number of "SCSI Disk Errors" increases at an unusually high rate, you could be about to loose a particular disk drive. The same can be said for network throughput when "Rx_Errors" do the same.

NOTE: A LeakyBucket can only monitor **counter** type statistics.

A LeakyBucket monitor is defined by its **bucketSize**,a **fillValue** and a **monitoringRate.** The bucket level (initially full) decreases over time by the same amount as its counter statistic increases. The monitoringRate control attribute determines how often the bucket level is replenished with the fillValue. If the bucket level ever equals 0 or less, i.e. the bucket becomes empty, then an exception event is generated.

A LeakyBucket is basically a rate monitor. If a counter statistic value increases on average by more than the **fillValue** every sampling interval (**monitoringRate**) over a long enough period to deplete the **bucketSize**, then an event is generated. The fillValue/monitoringRate can be described as an acceptable rate of increase for a counter statistic.

NOTE: All LeakyBucket configuration parameters except the type can be modified after a daemon monitor is created.

rmMonitorControl

The control structure specifies where and how often a monitor samples a statistic and reports an event. Attributes of the rmMonitorControl are listed in the figure below.

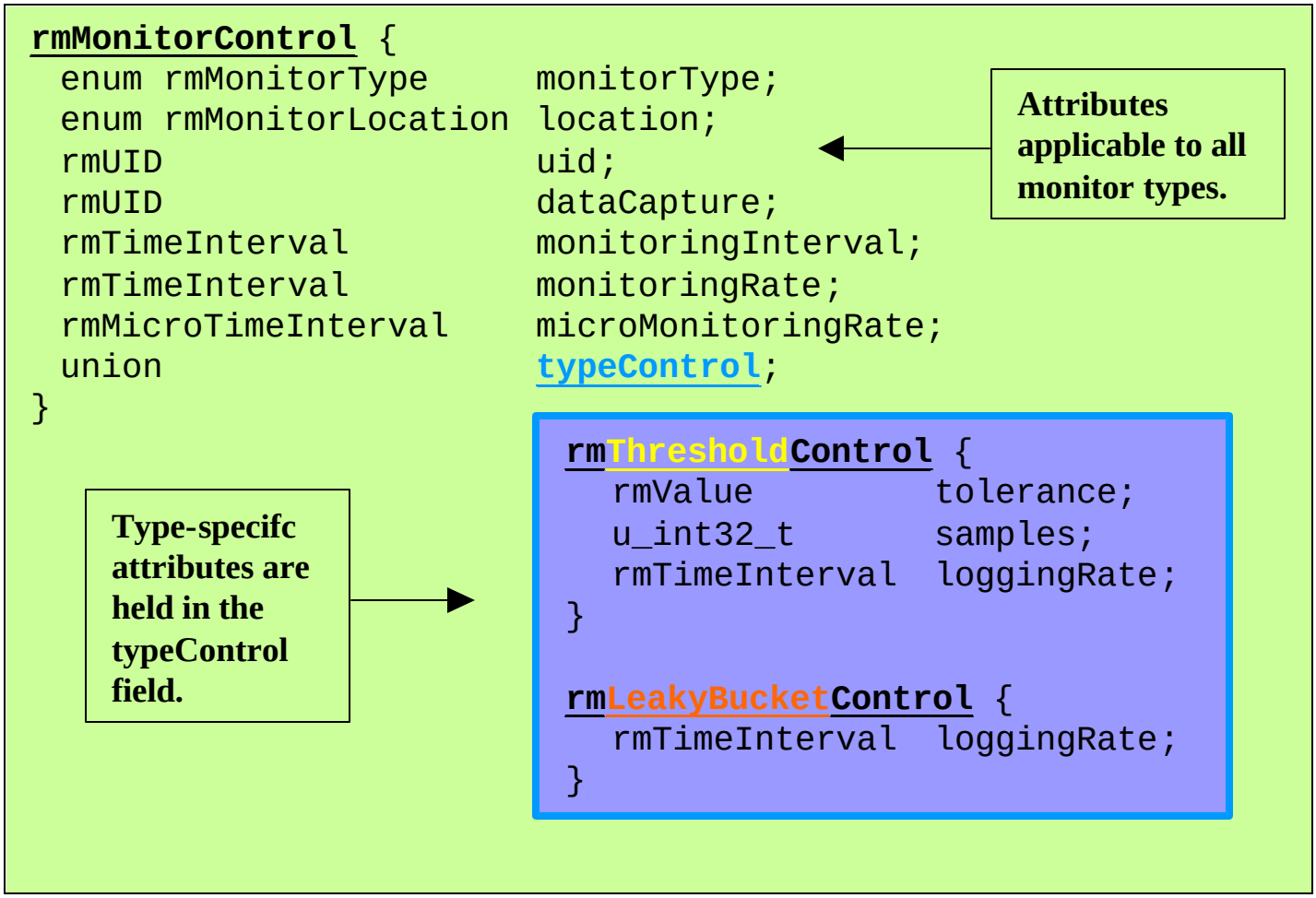

Figure 2 - rmMonitorControl attributes.

Meaning of the **rmMonitorControl** fields follows…

monitorType - specifies either **rmWatermark**, **rmThreshold**, or **rmLeakyBucket**
location - **rmInLine** (monitor created where the statistic is maintained)
rmDaemon (monitor created in the resourcemond process)
rmInLineDaemon (inline, if available, else daemon will be created)
uid - daemon uses this uid, if supplied, else it generates a new unique id.
dataCapture - enables historical data capture with the specified plug-in, if supplied.
monitoringInterval - duration (in seconds) of monitor life-span, (0 means last forever)
monitoringRate - minimum seconds between samples (60 = 1 sample per minute)
microMonitoringRate - micro-seconds between samples, added to monitoringRate
(daemon-side monitors do NOT support microMonitoringRate)
typeControl - holds either a **rmThresholdControl** or a **rmLeakyBucketControl.**

Meaning of the **rmThresholdControl** fields follows…

tolerance - tolerable +/- range for defining a statistical sample as “equal to”
to the **thresholdValue** found in **rmThresholdConfiguration**.
(Only used for **condition** of either **rmValueIsAt** or **rmValueIsNotAt**)

samples - number of consecutive readings required before changing state which implies either setting a new watermark or generating an event. (This attribute can help ignore noise within statistical samples)

loggingRate - minimum seconds between generating events (0 means log only one event, then assume **pauseNotification** state)

Meaning of the **rmLeakyBucketControl** fields follows…

loggingRate - minimum seconds between generating events (0 means log only one event, then assume **pauseNotification** state)

There is no control structure specific to watermark monitors.

rmMonitorConfiguration

The configuration structure holds details on how statistic samples are evaluated by a monitor and specifies the event severity of any exceptions that may be detected. Attributes of the rmMonitorConfiguration are listed in the figure below.

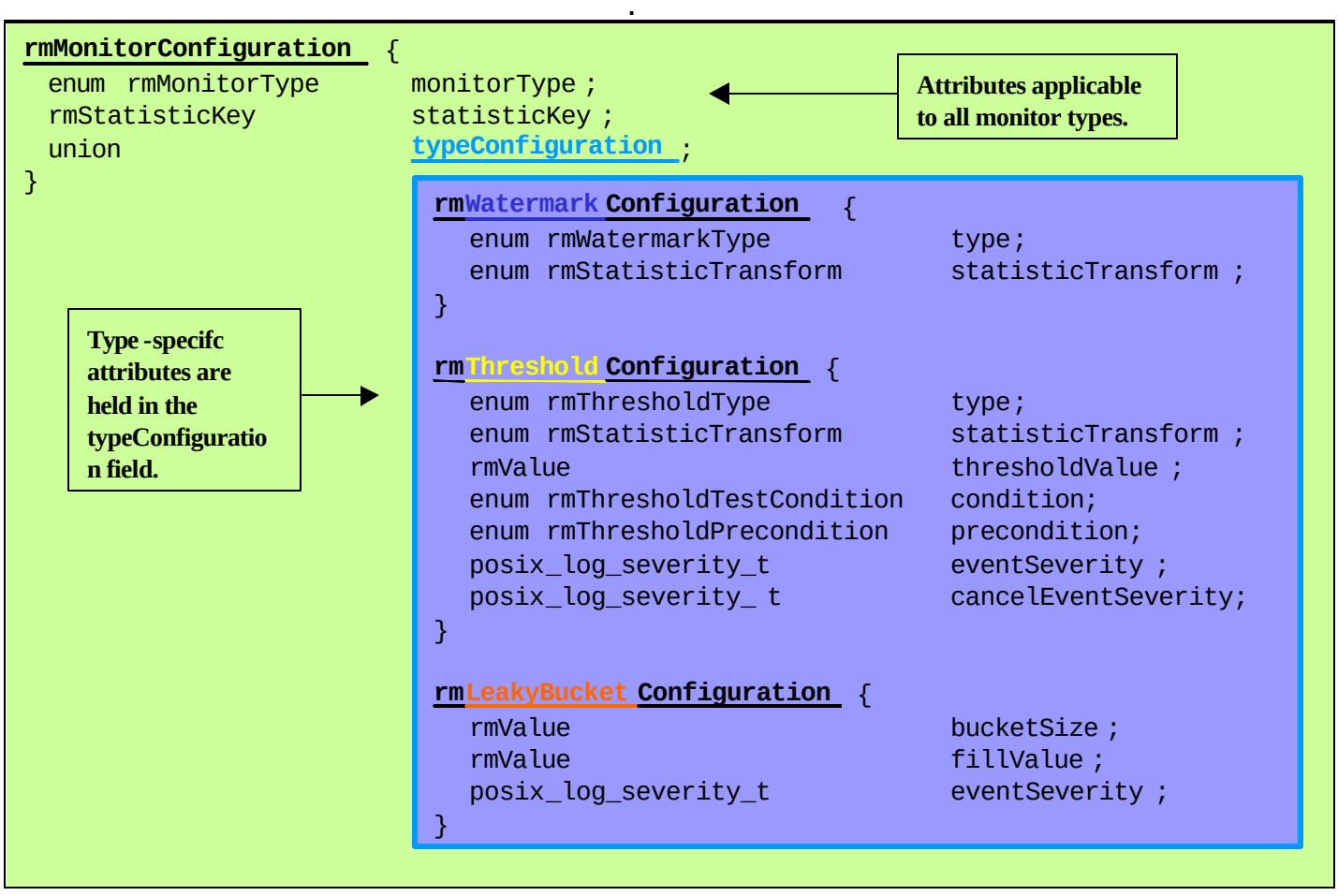

Figure 3 - rmMonitorConfiguration attributes

Meaning of the **rmMonitorConfiguration** fields follows…

monitorType - specifies either **rmWatermarking**,
rmThresholding,
or **rmLeakyBucket**

statisticKey - identifies the statistic to be monitorred
includes a 64-bit **SubsystemId,** a **ResourceId** and a **StatisticId**

typeConfiguration - points to either **rmWatermarkConfiguration,**
rmThresholdConfiguration,
or **rmLeakyBucketConfiguration.**

Meaning of the **rmWatermarkConfiguration** fields follows…

type - specifies either **rmHighWatermark, rmLowWatermark,**
or **rmDualWatermark**

statisticTransform - monitors an interpretation of a statistic as opposed to its raw value
can be either **rmChange, rmPercent**, or **rmPercentChange** where
rmChange monitors the largest increase of a HIGH watermark
or the largest decrease of a LOW watermark
rmPercent monitors percent (for statistics providing upper bound)
supported, ONLY if the upper bound is made available
using **rmGetUpperBound/ rmGetUpperBound** API's
rmPercent monitors percent change from one sample to the next

Meaning of the **rmThresholdConfiguration** fields follows…

type - specifies either **rmThreshold** or **rmBiDirectionalThreshold**

statisticTransform - monitors an interpretation of a statistic as opposed to its raw value can
be either **rmChange, rmPercent**, or **rmPercentChange** where
rmChange monitors the largest increase of a HIGH watermark
or the largest decrease of a LOW watermark
rmPercent monitors percent (for statistics providing upper bound)
ONLY supported, if the upper bound is made available
using **rmGetUpperBound/ rmGetUpperBound** API's
rmPercent monitors percent change from one sample to the next

thresholdValue - threshold level applied to the statistic or transform being monitored
Assumes the same data type as the monitored statistic, but is
assumed to be signed when the **rmChange** transform is used.
Valid threshold values are 0…100, when **rmPercent** is used

condition - test condition to compare the statistic value with the thresholdValue.

Can be…	TRUE if…
rmValueIsAbove	observed value > threshold value.
rmValueIsAtOrAbove	observed value >= threshold value.
rmValueIsAt	observed value == threshold value. (or within the control tolerance)
rmValueIsAtOrBelow	observed value <= threshold value.

	rmValueIsBelow	observed value < threshold value.
	rmValueIsNotAt	observed value != threshold value. (or NOT within the control tolerance)

precondition - specifies any condition that must be met before events are logged. Can be either **rmNoPrecondition** or **rmObserveGoodValue** that requires a sample that does not exceed the threshold condition. (rmObserveGoodValue can help ignore noise from early samples)

eventSeverity - specifies the POSIX severity level for exception events

Can be…	meaning…
LOG_EMERG	system is unusable
LOG_ALERT	action must be taken immediately
LOG_CRIT	critical conditions
LOG_ERR	error conditions
LOG_WARNING	warning conditions
LOG_NOTICE	normal but significant condition
LOG_INFO	informational
LOG_DEBUG	debug-level messages

cancelEventSeverity - specifies the POSIX severity level for exception canceling events
Uses the same values as eventSeverity
(Supported ONLY for **rmBiDirectionalThreshold**'s)

Meaning of the **rmLeakyBucketConfiguration** fields follows…

bucketSize - contains the maximum bucket level
fillValue - contains the periodic fill value
eventSeverity - specifies the POSIX severity level for exception events
Uses the same values as a threshold eventSeverity

Event Notification and Response

Once Threshold or LeakyBucket monitors are created and started, they have the potential for generating system events to report their exceptions. This section provides some brief insight into using the POSIX Event Log. For more in-depth PEL interface documentation go to **http://evlog.sourceforge.net.**

The **evlnofity** utility allows you to …

- Register a PEL query to detect when these events have occurred and
- Invoke a script in response to these events

The syntax for this command would look something like this…

```
evlnotify   --add "<path/scriptname> %recid% " \
            --filter "data contains \"Monitor_ID=<UUID_of_Monitor>\""
```

where **<path\scriptname>** is the fully qualified path to the response script.
%recid% is a parameter placeholder for the actual event record id to be passed to the response script at the time of execution.
<UUID_of_Monitor> is the UUID of the monitor being queried

For a C/C++ application to be notified directly, a new query must first be created and registered using the **`posix_log_query_create()`** function and the c/c++ callback/response function pointer must be associated with the new query using **`posix_log_notify_add().`**

Testing With Pseudo Events

You can generate a system event to test your response scripts with the following command...

```
/sbin/evlsend   --facility LOG_LOCAL5 \
                --type     494    \
                --severity ERR \
                -m "your message|Monitor_ID=<UUID_of_Monitor>|attr1=val| attr2=val|"
```

RM typically uses the **facility** code LOG_LOCAL5. The **type** is user-definable (we picked a random number for the example above). The **severity** can be any one of EMERG, ALERT, CRIT, ERR, WARNING, NOTICE, INFO, and DEBUG.

Respond to the event generated above with the following command...

```
/sbin/evlnotify  --add  "complete_path/respond.sh %recid%"
                 --filter  "data contains \" Monitor_ID=<UUID_of_Monitor>\""
```

Contents of **respond.sh** could be...

```
#!/bin/sh
   RecordString=`evlview --filter "recid=$1"`
su root -c "wall $RecordString"
```

This would print the event record string to all consoles on the system.

Query The Event Log

The response script or listener function may need to retrieve the event record in question for further investigation.

This is easily done with **evlview –filter "recid=$1"** in a shell script, but for C/C++ it requires a little more work. One way to do this quickly is described below…

First you need to get the record id with **`posix_log_siginfo_getrecid()`** and create a new query to find this record id with **`posix_log_query_create()`**. Then open the event log with

`posix_log_open()`, seek to the end (**`POSIX_LOG_SEEK_END`**) with **`posix_log_seek()`**, and then seek backwards (**`POSIX_LOG_SEEK_BACKWARD`**) with **`posix_log_seek()`** while querying for the record id. When the query returns, a call to **`posix_log_read()`** will give you the event record string.

Event Record Schema Formats

Once the event record string is in-hand, it can be parsed for further details of the event. The string may follow a number of different schema formats defined in **/usr/include/ResourceMonitor/MonitoringEventSchema.h**

One sample schema is shown below..

```
#define ME_ThresholdEventLogMessageSignedFormatWithNames  "%s\
|facility_id=2f737514-1d1b-4edb-9cd4-adffc433d56e\
|Monitor_ID=%s|Subsystem_ID=%s|Subsystem_Name=%s\
|Resource_ID=%u|Resource_Name=%s|Statistic_ID=%u|Statistic_Name=%s\
|Observed_Value=%d|Threshold_Value=%d|\n"
```

Additional Capabilities

Query and Modify Monitors

Sample Statistic Values

To sample current statistic values use either…

rmxml status –a

and look for the **<reading>** tag within the **<statisticStatus>** tags,

or use the C/C++ APIs rmGetCurrentValue() and Rmstatistic::GetCurrentValue();

Statistics can be either signed or unsigned, 32-bit or 64-bit values.

```
enum rmStatisticSize
{
 rmSizeU32 = 1,   ///< unsigned 32-bit value
 rmSizeS32,       ///< signed 32-bit value
 rmSizeU64,       ///< unsigned 64-bit value
 rmSizeS64       ///< signed 64-bit value
};
```

The meaning of the statistic value itself is more precisely interpreted by applying its scale. RM defines a variety of possible scales that a statistic might use.

```
enum rmStatisticScale
{
       rmScaleNone = 1,     ///< No units specified
       rmScaleByte,         ///< Bytes
       rmScaleKilobyte,     ///< Kilobytes (1024)
       rmScaleMegabyte,     ///< Megabytes (1024^2)
       mScaleGigabyte,      ///< Gigabytes (1024^3)
       rmScaleTerabyte,     ///< Terabytes (1024^4)
       rmScalePages,        ///< pages
       rmScaleJiffies,      ///< jiffies
       rmScaleNanosec,      ///< nanoseconds
       rmScaleMicrosec,     ///< microseconds
       rmScaleMillisec,     ///< milliseconds
       rmScaleSec,          ///< seconds
       rmScaleMin,          ///< minutes
       rmScaleHour,         ///< hours
       rmScaleDay,          ///< days
       mScaleWeek,          ///< weeks
       rmScaleMonth,        ///< months
       rmScaleYear          ///< years
};
```

Maintain Awareness of Resource Availability

It may be very important for your application to maintain a constant awareness of all available resource and when they might become unavailable. One method of doing this is to register a query for all Resource Monitor events and checki for events of two specific types.

1. ME_EVENT_SUBSYSTEM_CONFIGURATION_CHANGE (509)
2. ME_EVENT_RESOURCE_CONFIGURATION_CHANGE (510)

All events related to the Resource Monitor use a common **facility_id** attribute equal to **`2f737514-1d1b-4edb-9cd4-adffc433d56e.`** To register a query for all RM events using shell scripts use this command…

```
evlnotify  --add "<path/scriptname> %recid% " \
           --filter \
           "data contains \"facility_id=2f737514-1d1b-4edb-9cd4-adffc433d56e \""
```

The response script use for **<path/scriptname>** above can then parse the event record to check if **event_type=509** or **510**.

Using C/C++, a listener/response function can be assigned to all RM events like so…

```
sprintf( query_string, "data contains \"facility_id=%s\"", ME_UID_STRING);
report_events( NULL, query_string, Listener, 2);
```

See the sample code for more on the **report_events()** function. The **Listener**() function would have to retrieve the event record and compare its event_type to either …

ME_EVENT_SUBSYSTEM_CONFIGURATION_CHANGE

or

ME_EVENT_RESOURCE_CONFIGURATION_CHANGE.

Capture Historical Data

Selecting the best attributes for a monitor can be a difficult challenge, especially when little is known about the typical behavior of the targeted statistic. Capturing historical data on the statistic can be a useful tool for determining a more accurate threshold or LeakyBucket fillValue. The Resource Monitor currently provides two dataCapture libraries that record statistical samples to a file using either tab-delimited format or XML.

```
#define   TEXT_TABLE_DATACAPTURE_UUID_STRING  "9250f38b-e7b0-44b1-89c1-178ee59e281f"
#define   TEXT_XML_DATACAPTURE_UUID_STRING "7b2bd294-cdd1-49ae-b41f-e964e87c5370"
```

Historical data can be captured for a statistic simply by assigning a monitor to the statistic while also specifying a UUID for the dataCapture library to be used. Using XML, this UUID is set with the **configuration.dataCapture** element. Uusing C/C++, this UUID is set with the **monitorControl.dataCapture**.attribute.

The TEXT data capture library can be configured by modifying **\opt\resourcemon\etc\opt\resourcemon\textconfig**. This file contains the following attributes…

```
TEXTDataCaptureDirectory = /var/opt/resourcemon/data
TEXTMinimumFreeSpace  = 5
TEXTActiveMonitorLimit  = 0
TEXTFileAgeLimit  = 168
TEXTFileSizeLimit =  0
TEXTFileRollover  = false
```

Creating A Custom Subsystem

The subsystem choices provided with the Resource Monitor can be expanded by developing your own custom subsystem library and adding its path to the **/opt/resourcemon/rmtab** file.

A custom subsystem can be created by following five essential steps.

1) Obtain a universally unique id using the **uuidgen** command line utility

2) Compose meaningful Names and Descriptions for the subsystem, potential resources and for each statistic for which the subsystem will provide values.
3) Define all attributes for each statistic including a meaning description for units.
4) Provide the information derived above to the RM daemon via the **`m_subInfo`**, **`m_resourceInfo`**, and the **`m_statisticInfo[]`** structures.
5) Implement the **readValue()** method for supplying values for each statistic to the RM daemon.

A very basic template for writing your own subsystem is provided in the Appendix and can be found in the sample code provided with this document.

Resource Monitor Installation

Required Packages

The examples provided with this user guide require packages that are part of version 1.5 Carrier-Grade Linux Technology (CLT). CLT version 1.5 is based on the RedHat 7.3 kernel version 2.4.18. Download the latest CLT 1.5 packages from **http://www.sourceforge.net.**

TLT-dev-v1_5-build-xx.tar.gz (where **xx** is the latest build number)

This tar file contains over fifty (50) packages, of which only the following are required for RM:

kernel-2.4.18-xx.i386.rpm

evlog-1.4.0-1.i386.rpm

evlog-telco-1. 4.0-1.i386.rpm

resourcemonitord-1.5-1.i386.rpm

rmxml-1.0-1.i386.rpm

Additional Packages (for Telco Alarms and SNMP)

As explained in the introduction, Resource Monitor can be used for many purposes. One obvious use in any HA environment is to trigger alerts and alarms during critical system states. The following describes the CLT 1.5 packages required to trigger SNMP traps or to manipulate the front panel alarm LEDs (and rear external relays) of the Intel Carrier Grade TSRxT2 server platforms.

The remaining packages are **not** required to operate the provided sample code.

However, they are necessary, if you plan to ...

1. monitor ethernet statistics for the Intel e100 adapter,
2. trigger telco alarm LEDs on the TSRLT2/TSRMT2 (TSRxT2) front panel,
3. or trigger SNMP traps.

crms-0.1.0-1.i386.rpm (required for e100.o resources and other RM-instrumented drivers)

e100-telco-hard-2.4.18-TLT-SMP-2.0.30-x.i386.rpm (required for e100.o resources)

You only need to install these rpms (if required):

dmisp-1.0-6.i386.rpm (required for SNMP and telco alarms)

dmi2snmp-1.0-15.i386.rpm (required for SNMP)

isc-3.5.2-1.i386.rpm (required for telco alarms)

tam-1.1-1.i386.rpm (required for telco alarms)

Sanity Check (to Insure Proper Installation)

You should confirm that these processes are functioning properly...

...evlogd

...evlnotifyd
...evlactiond
...evlforward
...resourcemonitord
...tamd (for telco alarms support)
...snmptraps (for SNMP support)
and **lsmod** should list
...ipmi_comb (for telco alarms support)

Starting Resource Monitor

After installing the rpm, **resourcemonitord** is started automatically as it is for all subsequent reboots. Nothing additional is required to begin creating and managing monitors for the resources on your system. However, you may find some of these additional options helpful for tuning and debugging

Command Line Options

Modify **/etc/sysconfig/resourcemonitor** to set **resourcemonitord** options.

resourcemonitord [-d]* debug, executes as foreground process
(very verbose , with open stdout/stderr file descriptors).

[-t rmtab file] subsystem plug-ins are registered from this file
(Default: /etc/opt/resourcemon/rmtab)

[-p persist file] monitors are recreated from a persistent store.
(Default: /var/opt/resourcemon)

[-s socket] unsupported in this release.
(Default: /var/run/.RM-socket)

[-r Refresh Rate] frequency that RM checks for subsystem and resource changes.
(Default: 60 seconds)

[-m Monitoring Rate] frequency that RM CPU/memory utilization is checked/throttled.
(Default: 15 seconds)

[-c CPU limit] percent CPU utilization limit when RM begins to throttle itself.
(Default: 5 percent)

[-l memory limit] resident memory limit when RM begins to throttle itself.
(Default: 25 megabytes)

SIGNALS Understood by RM

You may easily send a signal to resourcemonitord using the following:

```
kill -SIGNAL `cat /var/run/resourcemonitord.pid`
```

resourcemonitord reacts to these signals:

SIGHUP causes resourcemonitord to exit normally.

SIGINT causes resourcemonitord to exit normally.

SIGTTIN causes an immediate read of rmtab, facilitating the installation of new subsystem library plug-ins.

SIGTTOU dumps some internal state details of the daemon into the file **/var/opt/resourcemon/rmState**.

Appendix - Interfaces for Resource Monitor

This appendix provides some brief configuration information on three possible development strategies for Resource Monitor. This information is by no means complete. Some knowledge of using either C, C++ or bash command shell is required. Additional documentation can be found at **/opt/resourcemon/doc/html/en/index.html**.

To reference known subsystems, their statistics and other attributes with more descriptive keywords, see the .DTD (Document Type Definition for XML scripts) and .H (header files for C/C++) that may be provided with each subsystem library.

Information common to both C and C++ environments follows.

The development environment for building an RM application will need **/usr/include/ResourceMonitor** in the INCLUDE path. **ResourceMonitor.h** is the only header required to include in your application source, but the other header files can be very helpful.

RM INCLUDE FILES: located in **/usr/include/ResourceMonitor**

For Application Developers:

ResourceMonitor.h	(required)
ResourceMonitorTypes.h	(RM structs, types, enums, etc.)
MonitoringEventSchema.h	(RM event record formats)
ResourceMonitorErrno.h	(RM error descriptions)
RM_Exception.h	
RM_FileSystemMonitor.h	(**FileSystem** subsystem macros)
RM_KernelMonitor.h	(**Kernel** subsystem macros)
RM_ProcessMonitor.h	(**ProcessMonitor** macros)
TextDataCapture.h	(**TextDataCapture** macros)

For Subsystem Developers:

IRMRegistration.h
ISubsystemMonitor.h
Monitor.h
Statistic.h

For Data Capture Developers:

StatisticalData.h

SHARED LIBRARIES: located in **/usr/lib/**

libResourceMonitor-1.5.so	(Resource Monitor API's)
libuuid.so	(universal unique id's for RM)
libevl.so	(POSIX Event Log API's)
libpthread.so	(POSIX threads to support PEL)
libnsl.so	(network socket layer)

RM applications are required to link with **"-levl -lnsl -lpthread -luuid -lResourceMonitor".**

Using Shell Scripts (XML)

XML Keywords:

Document Type Definition files describe entities referred to within the XML code.
DTD's can help make XML more readable by allowing the use of keywords.

For example, when referring to the SCSI subsystem we can use the keyword **&SCSI_SSID;** instead of the more cryptic unique id **cad21c0e-53eb-4261-8e2d-ee8aa5a49b09**.

NOTE: Don't forget the ever important semi-colon (**;**) needed as a sentinel to the keyword when writing your XML.

For more useful keywords, see these DTD files located in **/opt/resourcemon.**

subsystems	resourcemonitord
e100Subsystem.dtd	monitorData.dtd
fsSubsystem.dtd	monitorSpec.dtd
kernelSubsystem.dtd	monitorState.dtd
processSubsystem.dtd	rmCommon.dtd
scsiSubsystem.dtd	rmResources.dtd
	statisticStatus.dtd

A sample is shown below of the DTD for the SCSI Subsystem

```
<!-- SCSI resource monitor subsystem entities -->
<!ENTITY SCSI_SSID "cad21c0e-53eb-4261-8e2d-ee8aa5a49b09">
<!-- SCSI resource ids are dev numbers for local SCSI devices -->
<!ENTITY SCSI_TIMEOUTS          "0">
<!ENTITY SCSI_RESETS            "1">
<!ENTITY SCSI_PARITY_ERRORS     "2">
<!ENTITY SCSI_DISK_ERRORS       "3">
<!ENTITY SCSI_GROWN_DEFECTS     "4">
<!ENTITY SCSI_CAPACITY          "5">
<!ENTITY SCSI_TRANSIENT_ERRORS  "6">
<!ENTITY SCSI_USER_ERRORS       "7">
```

The rmxml Utility

rmxml discover [XML options] [<XML file>]

outputs vaild XML according to ***rmResources.dtd*** of available subsystems, resources, and statistics to stdout.

rmxml status [XML options] [-a | -i=uuid | <XML file>]

reads available statistics and outputs valid XML according to ***statisticStatus.dtd.***

-a read all available statistics.

-i=uuid uuid for the subsystem of statistics to read

rmxml create [XML options] [-g | -i=uuid] [<XML file>]

create monitor(s) from XML validated with ***monitorSpec.dtd.***

-g allow the daemon to generate the UUID for the monitor.

-i=uuid uuid for the monitor

XML file XML of monitor(s) to be created. if no file, input from stdin.

rmxml reconfigure [XML options] [<XML file>]

reconfigure existing monitor(s) from XML validated with ***monitorSpec.dtd.***

XML file XML of monitor(s) configured. if no file, input from stdin.

rmxml command [XML options] [-a | -i=uuid | <XML file>]

perform operation on existing monitor(s)

Commands:

state	outputs valid state XML according to ***monitorState.dtd.***
configuration	outputs valid configuration XML according to ***monitorSpec.dtd***.
delete	deletes monitor(s).
start	starts existing monitor(s).
stop	stops existing monitor(s).
reset	stops and resets existing monitor(s).
pauseNotification	pauses event log notifications for existing monitor(s).
resetNotification	restarts event log notifications for existing monitor(s).
-a	operate on all monitors.
-i=uuid	uuid for the monitor
XML file	identifies the monitor(s), validated with ***monitorSpec.dtd***. if no file, input from stdin.

XML options:

-u=xxx	Handle unrepresentable chars [fail | rep | ref*****].
-v=xxx	Validation scheme [always | never | auto*****].
-e	Expand Namespace Alias with URI's.
-x=XXX	Use a particular encoding for output (LATIN1*****).
-f	Enable full schema constraint checking. Defaults to off.
-p	Enable namespace-prefixes feature. Defaults to off.
-s	Disable schema processing. Defaults to on.
-?	Show this help.

*****Default if not provided explicitly.

The XML parser has intrinsic support for these encodings: UTF-8, USASCII, ISO8859-1, UTF-16[BL]E, UCS-4[BL]E,WINDOWS-1252, IBM1140, IBM037.

Using C

Miscelaneous

rmSessionHandle rmCreateConsumerSession
int rmCloseConsumerSession

Creation

rmHandle rmCreateMonitor
int rmDeleteMonitor

Discovery

int rmGetResourceMonitorVersion
size_t rmGetSubsystemCount
int rmGetAvailableSubsystems
int rmGetSubsystemInfo
rmString rmGetSubsystemDescription

int rmGetActiveResources
int rmGetResourceInfo
rmString rmGetResourceDescription
int rmGetResourceID

int rmGetAvailableStatistics
int rmGetStatisticInfo
rmString rmGetStatisticDescription
int rmGetStatisticID
int rmGetCurrentValue
int rmGetUpperBound

size_t rmGetMonitorCount
int rmGetMonitors
rmString rmGetMonitorDescription

Control

int rmResetCounterStatistic
rmHandle rmAccessMonitor
int rmGetMonitorControl
int rmSetMonitorControl
int rmGetMonitorConfiguration
int rmSetMonitorConfiguration
int rmStartMonitor
int rmStopMonitor
int rmResetMonitor
int rmPauseNotification
int rmResetNotification
int rmGetMonitorState
int rmGetMonitorInfo
int rmSetMonitorDescription

Using C++

miscelaneous

rmSessionHandle rmCreateConsumerSession
int rmCloseConsumerSession

class RMsession

rmSessionHandle getConsumerHandle
void setSessionHandle

class RMdiscovery

int GetActiveResources
int GetAvailableStatistics
int GetAvailableSubsystems
size_t GetMonitorCount
int GetMonitors
size_t GetSubsystemCount

class RMsubsystem

rmString GetSubsystemDescription
void GetSubsystemId
int GetSubsystemInfo

class RMresource

rmString GetResourceDescription
rmID GetResourceId
int GetResourceInfo

class RMstatistic

rmString GetStatisticDescription
rmID GetStatisticId
void GetStatisticId
int GetStatisticInfo
int GetUpperBound
int GetValue
int ResetCounterStatistic

class RMmonitor

int GetMonitorConfiguration
int GetMonitorControl
int GetMonitorInfo
int GetMonitorState
rmString GetMonitorDescription
void OnDestructionDeleteMonitor
int PauseNotification
int ResetMonitor
int ResetNotification
int SetMonitorDescription
int SetMonitorConfiguration
int SetMonitorControl
int StartMonitor
int StopMonitor

Appendix – Interfaces for the POSIX Event Log

Listed below are the event log interfaces that are used in the examples provided with this document. More information can be found at **http://evlog.sourceforge.net/linuxEvlog.html**

PEL Interfaces Using C

int posix_log_close

int posix_log_notify_add

int posix_log_open

int posix_log_query_create

int posix_log_query_destroy

int posix_log_read

int posix_log_seek

int posix_log_siginfo_getrecid

PEL Interfaces Using Shell Scripts

See the man pages for more complete descriptions of these commands.

evlnotify for creating and deleting PEL query notifications.

--add <notify-action-path>

--filter <filter>

--persistent

--delete <notify-id ...>

evlview for viewing event records from the POSIX event log file.

--filter <filter>

Appendix - Sample Shell Scripts (XML)

Main Shell Scripts

CreateMonitors.sh

```
#!/bin/sh
PWD=`pwd`
. common.sh

echo
echo
echo
echo This script is provided as sample code
echo for using the Resource Monitor "rmxml" utility.
echo
echo WARNING:  This program creates SAMPLE monitors
echo           on this server.
echo
echo
echo You must be 'root' to execute succesfully.
echo
echo Press CTRL-C to ABORT.
echo
read RESPONSE

# Create a THRESHOLD monitor for "used inodes"
# add a query response and start the monitor
#
NewMonitor_UUID=`rmxml create  Inodes_Threshold.xml | awk '{ print $3 }'`
evlnotify --add "$PWD/Inodes_Response.sh %recid%"  \
          --filter "data contains \"Monitor_ID=$NewMonitor_UUID\""
rmxml start -i=$NewMonitor_UUID
echo CREATED: `grep name Inodes_Threshold.xml | awk -Fname\> '{ print $2 }'`


# Create a LEAKYBUCKET monitor for "Disk Errors"
# add a query response and start the monitor
#
NewMonitor_UUID=`rmxml create DiskErrors_LeakyBucket.xml | awk '{ print $3 }'`
evlnotify --add "$PWD/DiskErrors_Response.sh %recid% MJR" \
          --filter "data contains \"Monitor_ID=$NewMonitor_UUID\""
rmxml start -i=$NewMonitor_UUID
echo CREATED: `grep name DiskErrors_LeakyBucket.xml | awk -Fname\> '{ print $2 }'`


# Create a WATERMARK monitor for "virtual memory size"
# and start the monitor (watermarks don't use queries)
#
NewMonitor_UUID=`rmxml create VirtualMemory_Watermark.xml | awk '{ print $3 }'`
rmxml start -i=$NewMonitor_UUID
echo CREATED: `grep name VirtualMemory_Watermark.xml | awk -Fname\> '{ print $2 }'`
```

```
# Create a THRESHOLD monitor for "percent used disk space"
# add a query response and start the monitor
#
NewMonitor_UUID=`rmxml create  PercentUsedSpace_Threshold.xml | awk '{ print $3 }'`
evlnotify --add "$PWD/PercentUsedSpace_Response.sh %recid%"  \
          --filter "data contains \"Monitor_ID=$NewMonitor_UUID\""
rmxml start -i=$NewMonitor_UUID
echo CREATED: `grep name PercentUsedSpace_Threshold.xml | awk -Fname\> '{ print $2
}'`


echo
echo
echo List SAMPLE monitors...
echo
showRM_SAMPLES

echo
echo
echo Press ENTER to modify the SAMPLE THRESHOLD above...
echo
read RESPONSE
# Modify the THRESHOLD created above from 95% to 90% Used Space...
# and change the name to reflect the new percentage.
#                                              new threshold value below       new
name value
rmxml configuration -i=$NewMonitor_UUID | sed 's/ue>95<\/rm/ue>90<\/rm/' | sed
's/95%/90%/' | rmxml reconfigure
#
# NOTE: The PEL query needs to be deleted and re-created
#       to reflect the new threshold value...
# ALSO NOTE:  If we were only to modify the query response script,
#             (without changing the monitor)
#             the PEL query would still need to be deleted and recreated
#             because the script is statically loaded when the query is created.
#
evlnotify --delete `evlnotify -l | grep dda82766-aba4-4ab7-b0ba-7578df6f52a3 | awk -
F: '{ print $1 }'`
evlnotify --add "$PWD/PercentUsedSpace_Response.sh %recid%"  \
          --filter "data contains \"Monitor_ID=$NewMonitor_UUID\""
echo THRESHOLD MODIFIED.
echo
echo
echo List SAMPLE monitors...
echo
showRM_SAMPLES

echo
echo Use "DeleteSamples.sh" to delete the SAMPLE monitors created by this script.
```

common.sh

```
#!/bin/sh
PATH=$PATH:/sbin:/opt/resourcemon_user_guide/scripts
#########################################################
# Common macros used with RM 1.5 User Guide sampe code
#

getIP(){
        ifconfig $1 2> /dev/null | grep "inet addr:" |
        awk -F":" ' { print $2 } ' | awk ' { print $1 } '
}


getEventRecord(){
        evlview -f "recid = $1"
}


getEventField(){
      echo $1 |  tr '|' '\n' | grep $2 | tr ',' '\n' | grep $2 |
      awk '{
             print substr( $0, length( paramName )+2)

           }' paramName=$2 paramDelim=$3
}


showRM_SAMPLES(){
      DUMPFILE=/var/opt/resourcemon/rmState
      # NOTE: Resource Monitor does NOT support Content of the DUMPFILE.
      #       This file should not be relied upon for any reliable format.
      #       resourcemon uses 'dump' paramater for debug purposes only.
      /etc/rc.d/init.d/resourcemon dump
      while [ ! -f $DUMPFILE ] ; do
           sleep 4
      done
      cat $DUMPFILE  | grep SAMPLE
}


deleteSamples() {
    for i in `rmd dump | grep SAMPLE: | awk '{ print $2 }'`; do
        rmxml delete -i=$i
    done
}
```

DeleteMonitors.sh

```
#!/bin/sh
PWD=`pwd`
. common.sh

deleteSamples
```

Monitor Spec Files

Inodes_Threshold.xml

```
<?xml version = "1.0" standalone="no"?>
<!DOCTYPE MonitorSpec SYSTEM "/opt/resourcemon/monitorSpec.dtd">
<MonitorSpec>
<monitorSpec>

<!--    This monitor defines a threshold for "used inodes" exceeding 59
#       for the "/boot" (2050) resource of the FileSystem subsystem.
#       Exceptions are reported every 10 seconds.
#
#                                Parameters values are aligned in
#                                the column below...
#
-->
    <monitor> <name>SAMPLE: THRESHOLD for Used Inodes > 59 on /boot</name>
              <statisticKey>
                    <subsystem>  <UUID>&FileSystemSSID;</UUID>
</subsystem>
                     <resource>  <rmID>  2050                    </rmID></resource>
                    <statistic>
<rmID>&FILESYSTEM_USED_INODES_ID;</rmID></statistic>
              </statisticKey>
    </monitor>

    <configuration               type="thresholding"
                                 location="daemon">
                      <interval>  0                               </interval>
                          <rate>  1                               </rate>
      <thresholdConfiguration    type="threshold"
                                 condition="above" >
         <threshold>  <rmValue>  59                               </rmValue>
</threshold>
         <tolerance>  <rmValue>  0                                </rmValue>
</tolerance>
         <thresholdEvent>        <event severity="critical" />    </thresholdEvent>
         <samples>               1                                </samples>
         <loggingRate>           10                               </loggingRate>
      </thresholdConfiguration>
<!--
# To enable HISTORICAL DATA CAPTURE, insert one of the following lines…
#                 <dataCapture>  &TEXT_TABLE_DCID;             </<dataCapture>
#                 <dataCapture>  &TEXT_XML_DCID;               </<dataCapture>
-->
    </configuration>
</monitorSpec>

<!--  More monitorSpec's can follow for defining multiple monitors.
#
#   <monitorSpec>
#   .
#   .
#   .
```

```
#   </monitorSpec>
#
-->

</MonitorSpec>
```

DiskErrors_LeakyBucket.xml

```
<?xml version = "1.0" standalone="no"?>
<!DOCTYPE MonitorSpec SYSTEM "/opt/resourcemon/monitorSpec.dtd">
<MonitorSpec>
<monitorSpec>

<!--    This monitor defines a leakybucket for "SCSI Disk Errors" statistic (3)
#       exceeding a rate of 5 per day for the "/dev/sda5" resource (0)
#       of the FileSystem subsystem.
#       Exception events are created every 60 seconds.
#
#                                Parameters values are aligned in
#                                the column below...
#
-->
   <monitor> <name>SAMPLE: LEAKYBUCKET for SCSI Disk Errors > 5 per day on
/dev/sda5</name>
             <statisticKey>
                    <subsystem>  <UUID>&SCSI_SSID;</UUID>       </subsystem>
                     <resource>  <rmID>  0               </rmID></resource>
                    <statistic>  <rmID>&SCSI_DISK_ERRORS;</rmID></statistic>
             </statisticKey>
   </monitor>

    <configuration               type="leakyBucket"
                                 location="daemon">
                      <interval> 0                            </interval>
                          <rate> 86400                        </rate>
     <leakyBucketConfiguration>
         <bucketSize> <rmValue> 5                 </rmValue> </bucketSize>
          <fillValue> <rmValue> 4                 </rmValue> </fillValue>
             <leakyBucketEvent> <event severity="emergency"/> </leakyBucketEvent>
                  <loggingRate> 60                            </loggingRate>
    </leakyBucketConfiguration>
<!--
# To enable HISTORICAL DATA CAPTURE, insert one of the following lines…
#                <dataCapture> &TEXT_TABLE_DCID;             </<dataCapture>
#                <dataCapture> &TEXT_XML_DCID;               </<dataCapture>
-->
    </configuration>
</monitorSpec>
<!--  More monitorSpec's can follow for defining multiple monitors.
#
#   <monitorSpec>
#   .
#   .
#   .
#   </monitorSpec>
#
```

```
-->

</MonitorSpec>
```

VirtualMemory_Watermark.xml

```
<?xml version = "1.0" standalone="no"?>
<!DOCTYPE MonitorSpec SYSTEM "/opt/resourcemon/monitorSpec.dtd">
<MonitorSpec>
<monitorSpec>

<!--    This monitor tracks a High Watermark for process init (pid=1)
#       referred to as resource 1 of the ProcessMonitor subsystem
#
#                          Parameters values are aligned in
#                          the column below...
#
-->
 <monitor> <name>SAMPLE: WATERMARK for Virtual Memory - process id 1 "init"</name>
           <description>Virtual Memory Size High Watermark for process (1) init
</description>
           <creator>500</creator>
           <statisticKey>
              <subsystem> <UUID>&ProcessSSID;</UUID> </subsystem>
               <resource> <rmID> 1            </rmID></resource>
              <statistic> <rmID>&STAT_vsize_ID;</rmID></statistic>
           </statisticKey>
 </monitor>
 <configuration           type="watermarking"
                          location="daemon">
              <interval>  0                   </interval>
                  <rate>  1                   </rate>
             <microrate>  1                   </microrate>
 <watermarkConfiguration  type="high"
                          transform="none"/>
<!--
# To enable HISTORICAL DATA CAPTURE, insert one of the following lines…
#                 <dataCapture>  &TEXT_TABLE_DCID;             </<dataCapture>
#                 <dataCapture>  &TEXT_XML_DCID;               </<dataCapture>
-->
 </configuration>
</monitorSpec>

<!--  More monitorSpec's can follow for defining multiple monitors.
#
#   <monitorSpec>
#   .
#   .
#   .
#   </monitorSpec>
#
-->

</MonitorSpec>
```

PercentUsedSpace_Threshold.xml

```
<?xml version = "1.0" standalone="no"?>
<!DOCTYPE MonitorSpec SYSTEM "/opt/resourcemon/monitorSpec.dtd">
<MonitorSpec>
<monitorSpec>

<!--    This monitor defines a threshold for "percent used disk space"
#       exceeding 95
#       for the "/" (2053) resource of the FileSystem subsystem.
#       The statistic is sampled every 5 minutes (300 seconds)
#       Exceptions are reported every 2 minutes (120 seconds)
#
#                               Parameters values are aligned in
#                               the column below...
-->
    <monitor> <name>SAMPLE: THRESHOLD for Percent Used Disk > 95% on "/"</name>
              <statisticKey>
                    <subsystem>  <UUID>&FileSystemSSID;</UUID>
</subsystem>
                     <resource>  <rmID>  2053
</rmID></resource>
                    <statistic>
<rmID>&FILESYSTEM_PERCENT_USED_SPACE_ID;</rmID></statistic>
             </statisticKey>
    </monitor>

    <configuration              type="thresholding"
                                location="daemon">
                     <interval>  0                               </interval>
                         <rate>  300                             </rate>
      <thresholdConfiguration   type="bidirectional"
                                condition="atOrAbove" >
         <threshold>  <rmValue>  35                              </rmValue>
</threshold>
         <tolerance>  <rmValue>  0                               </rmValue>
</tolerance>
         <thresholdEvent>       <event severity="warning" />    </thresholdEvent>
         <samples>              1                               </samples>
         <loggingRate>          120                             </loggingRate>
      </thresholdConfiguration>
<!--
# To enable HISTORICAL DATA CAPTURE, insert one of the following lines…
#                 <dataCapture>  &TEXT_TABLE_DCID;            </<dataCapture>
#                 <dataCapture>  &TEXT_XML_DCID;              </<dataCapture>
-->
    </configuration>
</monitorSpec>

<!--  More monitorSpec's can follow for defining multiple monitors.
#   <monitorSpec>
#   .
#   .
#   .
#   </monitorSpec>
-->
```

```
</MonitorSpec>
```

Event Reponse Scripts

Inodes_Response.sh

```
#!/bin/sh
#########################################################
# InodesThreshold_Response
#             invoked by evlactiond
#             for every exception from the monitor
#             created by InodesThreshold_LeakyBucket.xml.
#########################################################
. common.sh
RecordID=$1
Alarm=$2
PWR_Alarm=$3

EventRecord=`getEventRecord $RecordID`
CancelEvent=`getEventField "$EventRecord" "event_type=" ","`
EventSeverity=`getEventField "$EventRecord" "severity=" ","`
EventTime=`getEventField "$EventRecord" "time=" ","`
EventTitle=`getEventField "$EventRecord" "processor=1" "|"`
StatName=`getEventField "$EventRecord" "stat_name" "|"`
ObservedVal=`getEventField "$EventRecord" "observed_val" "|"`
Message="$EventTime : $StatName=$ObservedVal"

   if [ "X$EventRecord" = "X" ] ; then
      EventTitle="evlview ERROR"
      Message="Event Record $RecordID Not Found"
      EventSeverity="QWARNING"
   fi
   wall  "$EventSeverity - $EventTitle: $Message"  &

# TODO - Add more appropriate actions here

if [ "$CancelEvent" = "500" ] ; then
     Alarm=0
fi
if [ "$Alarm" != "" ] ; then
     su root -c "tap $Alarm $PWR_Alarm" &
fi
```

DiskErrors_Response.sh

```
#!/bin/sh
#########################################################
# DiskErrors_Response
#             invoked by evlactiond
#             for every exception from the monitor
#             created by DiskErrors_LeakyBucket.xml.
#########################################################
. common.sh
RecordID=$1
Alarm=$2
PWR_Alarm=$3
```

```
Button0=OK
EventRecord=`getEventRecord $RecordID`
# LeakBuckets cannot be biDirectional, therefore have no CancelEvent
#CancelEvent=`getEventField  "$EventRecord" "event_type=" ","`
EventTime=`getEventField     "$EventRecord" "time=" ","`
StatName=`getEventField      "$EventRecord" "stat_name" "|"`
ObservedVal=`getEventField   "$EventRecord" "observed_val" "|"`
EventSeverity=`getEventField "$EventRecord" "severity=" ","`
EventTitle=`getEventField    "$EventRecord" "processor=1" "|"`
Message="$EventTime : $StatName=$ObservedVal"

   if [ "X$EventRecord" = "X" ] ; then
      EventSeverity="QWARNING"
      EventTitle="evlview ERROR"
      Message="Event Record $RecordID Not Found"
   fi
wall  "$EventSeverity - $EventTitle: $Message"  &

#  TODO – Add more appropriate actions here…
   if [ "$Alarm" != "" ] ; then
      su root -c "tap $Alarm $PWR_Alarm" &
   fi
```

PercentUsedSpace_Response.sh

```
#!/bin/sh
#########################################################
# PercentUsedThreshold_Response
#             invoked by evlactiond
#             for every exception from the monitor
#             created by PercentUsedThreshold_LeakyBucket.xml.
#########################################################
. common.sh
RecordID=$1
Alarm=$2
PWR_Alarm=$3

EventRecord=`getEventRecord $RecordID`
CancelEvent=`getEventField "$EventRecord" "event_type=" ","`
EventSeverity=`getEventField "$EventRecord" "severity=" ","`
EventTime=`getEventField "$EventRecord" "time=" ","`
EventTitle=`getEventField "$EventRecord" "processor=1" "|"`
StatName=`getEventField "$EventRecord" "stat_name" "|"`
ObservedVal=`getEventField "$EventRecord" "observed_val" "|"`
Message="$EventTime : $StatName=$ObservedVal"
   if [ "X$EventRecord" = "X" ] ; then
      EventTitle="evlview ERROR"
      Message="Event Record $RecordID Not Found"
      EventSeverity="QWARNING"
   fi
wall  "$EventSeverity - $EventTitle: $Message"  &
# TODO - Add more appropriate actions here
if [ "$CancelEvent" = "500" ] ; then
     Alarm=0
fi
if [ "$Alarm" != "" ] ; then
```

```
     su root -c "tap $Alarm $PWR_Alarm" &
fi
```

Appendix – Sample C

```
/***************************************************************************
                          cSampleApp.c  -  description
                             -------------------
    Create               : Mon Jun 17 2002
    Copyright            : (C) 2002 by Zhu Yi
    Email                : yi.zhu@Intel.com
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software you can redistribute it and/or modify   *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; either version 2 of the License, or     *
 *   (at your option) any later version.                                   *
 *                                                                         *
 ***************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <posix_evlog.h>
#include <ResourceMonitor.h>
#include <RM_KernelMonitor.h>
#include <RM_FileSystemMonitor.h>
#include <MonitoringEventSchema.h>

#define  PrintError(format, arg...) fprintf(stderr, "Error %s: "format"\n", ##arg)
#define ErrorStr strerror(errno)


const char * GetStringbyUid (rmUID uid)
{
  static char uuid[36];

  /* Get the UUID string */
  uuid_unparse (uid, uuid);

  return uuid;
}


rmString  GetStatDesc (rmUID uid, rmID id)
{
  rmString  statDesc;

  statDesc = rmGetStatisticDescription (uid, id, rmShortDescription);
  if ( statDesc == NULL ) {
    if ( errno == ENXIO )
      PrintError ("Subsystem ID [%s] or Resource ID [%d] is not registered "
            "with the resource monitoring facility",
            "rmGetStatisticDescription", GetStringbyUid (uid), id);
```

```
    else
      PrintError ("%s", "rmGetStatisticDescription", ErrorStr);
    return NULL;
  }
  return statDesc;
}


void PrintStatDesc (rmUID uid, rmID id)
{
  rmString  statDesc;

  if ( (statDesc = GetStatDesc (uid, id)) ) {
    printf ("\t\tDESCRIPTION = [%s]\n", statDesc);
    free (statDesc);
    return;
  }
  exit (1);
}


int  PrintResDesc (rmUID uid, rmID id)
{
  rmString  resDesc;

  resDesc = rmGetResourceDescription (uid, id, rmShortDescription);
  if ( resDesc == NULL ) {
    if (errno == ENXIO)
      PrintError ("Subsystem ID [%s] or Resource ID [%d] is not registered "
            "with the resource monitoring facility",
            "rmGetResourceDescription", GetStringbyUid (uid), id);
    else
      PrintError ("%s", "rmGetResourceDescription", ErrorStr);
    exit (1);
  }
  printf ("\t\tDESCRIPTION = [%s]\n", resDesc);
  free (resDesc);

  return 0;
}


int  PrintSubSysDesc (rmUID uid)
{
  rmString  subDesc;

  subDesc = rmGetSubsystemDescription (uid, rmShortDescription);
  if ( subDesc == NULL ) {
    if (errno == ENXIO)
      PrintError ("%s is not registered with the resource monitoring facility",
            "rmShortDescription", GetStringbyUid (uid));
    else
      PrintError ("%s", "rmShortDescription", ErrorStr);
    exit (1);
  }
  printf ("\tDESCRIPTION = [%s]\n", subDesc);
  free (subDesc);
```

```
  return 0;
}


void PrintCurrentValue (rmStatisticId statKey)
{
  rmValue   value;
  int  retval;

  retval = rmGetCurrentValue (statKey, &value);
  if ( retval ) {
    if (retval == ENXIO)
      PrintError ("Any element of rmStatisticId is not registered with the "
            "resource monitoring facility", "rmGetCurrentValue");
    else
      PrintError ("%s", "rmGetCurrentValue", ErrorStr);
    exit (1);
  }
  printf ("Current Value for \"%s\" is: [%u]\n",
    GetStatDesc (statKey.SubsystemId, statKey.StatisticId), value.rmValueU32);
}


void discover_resource ()
{
  size_t iSub, iRes, iStat;
  size_t nSubsystem;
  int  retval;

  rmSubsystemInfo  *subBuffer, *pSub;
  rmResourceInfo*resBuffer, *pRes;
  rmStatisticInfo  *statBuffer, *pStat;


  /* Get total subsystem count in the system */
  nSubsystem = rmGetSubsystemCount ();
  printf ("SUBSYSTEM COUNT = [%d]\n", nSubsystem);

  /* Create a buffer to hold all the subsystem information */
  if ((subBuffer = (rmSubsystemInfo *) malloc (sizeof (rmSubsystemInfo) *
nSubsystem)) == NULL) {
    PrintError ("[%s, %d] malloc failed", "malloc", __FILE__, __LINE__);
    exit (1);
  }

  /* Get all the subsystem information */
  retval = rmGetAvailableSubsystems (subBuffer, nSubsystem);
  if (retval == EAGAIN) {
    PrintError ("size %d is not big enough", "rmGetAvailableSubsystems",
nSubsystem);
    exit (1);
  } else if (retval) {
    PrintError ("%s", "rmGetAvailableSubsystems", ErrorStr);
    exit (1);
  }
```

```
  /* For all the subsystems in this system */
  for (iSub = 0, pSub = subBuffer; iSub < nSubsystem; iSub++, pSub++) {

    const char * uuid;

    uuid = GetStringbyUid (pSub->id);
    PrintSubSysDesc (pSub->id);
    printf ("\tSUBSYSTEM UUID = [%s]\n", uuid);
    printf ("\tRESOURCE NUMBER = [%d]\n", pSub->noResources);
    printf ("\tSTATISTIC NUMBER = [%d]\n", pSub->noStatistics);


    resBuffer = (rmResourceInfo *) malloc (sizeof (rmResourceInfo) * pSub-
>noResources);
    if (resBuffer == NULL) {
      PrintError ("[%s, %d] malloc failed", "malloc", __FILE__, __LINE__);
      exit (1);
    }
    /* Get all the resources in this subsystem */
    if ((retval = rmGetActiveResources (pSub->id, resBuffer, pSub->noResources)) !=
0) {
      switch (retval) {
        case ENXIO:
          PrintError ("Subsystem ID [%s] not found",
                "rmGetActiveResources", uuid);
          break;
        case ENODATA:
          PrintError ("No resources are available",
                "rmGetActiveResources");
          break;
        case EAGAIN:
          PrintError ("A larger buffer is needed to retrieve them all",
                "rmGetActiveResources");
          break;
        default:
          PrintError ("%s", "rmGetActiveResources", ErrorStr);
      }
      exit (1);
    }

    /* For all the resources in this subsystem */
    for (iRes = 0, pRes = resBuffer; iRes < pSub->noResources; iRes++, pRes++) {
      printf ("\t\tRESOURCE ID = [%d]\n", pRes->id);
      PrintResDesc (pRes->SubsystemId, pRes->id);
    }

    /* Get all the statistics for this resource */
    statBuffer = (rmStatisticInfo *) malloc (sizeof (rmStatisticInfo) * pSub-
>noStatistics);
    if (statBuffer == NULL) {
      PrintError ("[%s, %d] malloc failed", "malloc", __FILE__, __LINE__);
      exit (1);
    }

    retval = rmGetAvailableStatistics (pSub->id, statBuffer, pSub->noStatistics);
    if (retval) {
      switch (retval) {
```

```
          case ENXIO:
            PrintError ("Subsystem ID [%s] not found",
                  "rmGetAvailableStatistics", uuid);
            break;
          case ENODATA:
            PrintError ("No resources are available",
                  "rmGetAvailableStatistics");
            break;
          case EAGAIN:
            PrintError ("A larger buffer is needed to retrieve them all",
                  "rmGetAvailableStatistics");
            break;
          default:
            PrintError ("%s", "rmGetAvailableStatistics", ErrorStr);
        }
        exit (1);
      }

      /* For all the resources in this subsystem */
      for (iStat = 0, pStat = statBuffer; iStat < pSub->noStatistics; iStat++,
pStat++) {
        printf ("\t\tSTATISTICS ID = [%d]\n", pStat->id);
        PrintStatDesc (pStat->SubsystemId, pStat->id);
        printf ("\t\tSTATISTICS TYPE = [%d]\n", pStat->type);
      }
    } /* End for all the subsystems in this system */

}


void discover_monitors ()
{
    size_t    nMonitor, i;
    rmMonitorInfo *monBuffer, *pMonitor;

    nMonitor  = rmGetMonitorCount ();
    monBuffer = (rmMonitorInfo *) malloc (sizeof (rmMonitorInfo) * nMonitor);

    if (rmGetMonitors (monBuffer, &nMonitor)) {
      if ( nMonitor == 0 )
        printf ("No Available Monitors\n");
      else {
        PrintError ("%s", "rmGetMonitors", ErrorStr);
        exit (1);
      }
    }
    for (i = 0, pMonitor = monBuffer; i < nMonitor; i++, pMonitor++) {

      rmHandle  p;
      const char *  uuid;

      uuid = GetStringbyUid (pMonitor->uid);
      printf ("Monitor ID = %s\n", uuid);
      printf ("uid = %d\n", pMonitor->creator);
      p = rmAccessMonitor (uuid);
      if ( !p ) {
        PrintError ("%s", "rmAccessMonitor", ErrorStr);
```

```
      continue;
    }
    if ( rmDeleteMonitor (p) )
      PrintError ("%s", "rmDeleteMonitor", ErrorStr);
  }
}


void myEventResponse (const char *event_string)
{
  printf ("Event Message is: %s\n", event_string);
  printf ("You can also do your action here\n");

      /* TODO - parse the event_string here for key
  *        information to act upon, such as event_type,
  *        severity, Subsystem_Name, Resource_Name
  *        Statistic_Name, Observed_Value, Threshold_Value,
  *        etc...
  *        (more fields found in MonitoringEventSchema.h)
  *
  * NOTE:  An event_type of ME_EVENT_CANCELED denotes that
  *        a bidirectional threshold is no longer exceeded,
  *        i.e. the statistic has returned to normal range.
  */

  fflush (stdout);
}


int  getEventRecord (posix_log_recid_t recid, char * event_string)
{
  char recid_query_string[POSIX_LOG_MEMSTR_MAXLEN];
  char error_string[POSIX_LOG_MEMSTR_MAXLEN];
  int  ret;

  posix_log_query_t  recid_posix_query;
  posix_logd_t     logdes;
  struct posix_log_entry  myentry, *entry;


  strcpy (event_string, "No Event String Yet");

  /* Searching the event log for record where id= recid */
  sprintf (recid_query_string, "recid = %d", recid);
  if( posix_log_query_create (recid_query_string, POSIX_LOG_PRPS_SEEK,
             &recid_posix_query, error_string, POSIX_LOG_MEMSTR_MAXLEN) ) {
    PrintError ("Could not create query for recid: %d  %s.",
"posix_log_query_create",
           recid, error_string);
    exit (1);
  }

  /* Let's get the entire event log entry */
  entry = &myentry;
  if ( !posix_log_open (&logdes, NULL) ) {

      /* Just read from bottom and get mesg. This is to be used
```

```
     with automated tests or faster response demos
     Register the query string and check syntax
         */
     errno = 0;
     if ( !(ret = posix_log_seek (logdes, NULL, POSIX_LOG_SEEK_END)) ) {

       if ( !(ret = posix_log_seek (logdes, &recid_posix_query,
POSIX_LOG_SEEK_BACKWARD)) ) {

         if ( !(ret = posix_log_read (logdes, entry, event_string,
                    POSIX_LOG_ENTRY_MAXLEN - 1)) ) {

           printf ("Found RECID: %d  MESG: %s\n",
             entry->log_recid, event_string);
           return 0;

         } else {
           PrintError ("FAILED to find recid=%d, err=%d, errno=%d."
                 "Attempt to process this event record: \"%s\"",
                 "posix_log_read", recid, (unsigned int)ret, errno,
                 event_string);
           return -1;
         }
       } else {
         sleep (1);
         errno = 0;
         if ( !(ret = posix_log_seek (logdes, NULL, POSIX_LOG_SEEK_END)) ) {

           if ( !(ret = posix_log_seek (logdes, &recid_posix_query,
                       POSIX_LOG_SEEK_BACKWARD)) ) {

             if ( !(ret = posix_log_read (logdes, entry, event_string,
                         POSIX_LOG_ENTRY_MAXLEN - 1)) ) {

               printf ("SUCCESS: FIRST backward seek FAILED!"
                 "SECOND seek found RECID: %d MESG: %s\n",
                        entry->log_recid, event_string);
             } else {
               PrintError ("First backward seek FAILED!"
                      "SECOND seekd FAILED to find"
                      "recid=%d, err=%d, errno=%d. Attempt "
                      "to process this event record: \"%s\"",
                      "posix_log_read", recid,
                      (unsigned int)ret, errno, event_string);
               return -1;
             }
           } else {
             PrintError ("First backward seek FAILED! SECOND seekd"
                    "FAILED to find recid=%d, err=%d, errno=%d."
                    "Attempt to process this event record: \"%s\"",
                    "posix_log_seek", recid, ret, errno,
                    event_string);
             return -1;
           }
         } else {
           PrintError ("Could not seek AGAIN to end of log file."
                 "ret=%d, errno=%d", "posix_log_seek", ret, errno);
```

```
            return -1;
          }
        }
        if ( posix_log_query_destroy( &recid_posix_query) )
          PrintError ("Failed for recid_posix_query", "posix_log_query_destroy");

     } else {
        PrintError ("Could not seek to end of log file. ret=%d, errno=%d",
              "posix_log_seek", ret, errno);
        return -1;
     }
     posix_log_close (logdes);
   } else {
     PrintError ("Could not open the event log file. errno=%d.", "posix_log_open",
errno);
     return -1;
   }

   fflush (stdout);
   return 0;
} /* getEventRecord() */


/*  A prior call to registerListener() uses posix_log_notify_add()
 *  to create a query and register this function as the callback.
 *  If this query matches any incoming event record, then this
 *  eventListener() function will be called.
 *
 *  This function responds to events related to
 *  Threshold and LeakyBuckets exceptions.
 */
void eventListener (int signo, siginfo_t * info, void * ignored)
{
   char      incoming_event_msg_string[POSIX_LOG_ENTRY_MAXLEN+128];
   sigset_t    old_sig, nold;
   posix_log_recid_t  recid;

   sigfillset (&nold);
   sigdelset (&nold, SIGINT);
   sigdelset (&nold, SIGQUIT);
   sigprocmask (SIG_SETMASK, &nold, &old_sig);

   /* Get the recid of the event that caused this function call. */
   posix_log_siginfo_getrecid (info, ignored, &recid);

   if ( !getEventRecord (recid, incoming_event_msg_string) )
        /* TODO - a fork() is more appropriate here, because
     we should not waste time inside the eventListener() */
     myEventResponse (incoming_event_msg_string);
   else
     PrintError ("Can't find offending RECID: %d", "getEventRecord", recid);

   sigprocmask (SIG_SETMASK, &old_sig, NULL);
}  /* eventListener() */
```

```
int  registerListener (rmHandle *newmonitor, void (* myListener) (int, siginfo_t *,
void *),
         char cmdAction[], int signo_offset)
{
   struct sigevent    mynotification;   //  Remember the monitor's query_handle
   posix_log_notify_t newquery_handle;  //  and its query so we can stuff them
   posix_log_query_t  newposix_query;   //  into the Monitorset to use later.
   rmMonitorInfo    mInfo;
   static struct sigaction SigRTAction;

   char error_string[POSIX_LOG_MEMSTR_MAXLEN];
   char newquery_string[POSIX_LOG_MEMSTR_MAXLEN];
   char MonIDString[MAX_GUID_STRLEN];
   char cmdString[POSIX_LOG_MEMSTR_MAXLEN];


   if ( rmGetMonitorInfo (newmonitor, &mInfo) ) {
     PrintError ("%s", "rmGetMonitorInfo", ErrorStr);
     return -1;
   }
   uuid_unparse (mInfo.uid, MonIDString);
   sprintf (newquery_string, "data contains \"Monitor_ID=%s\"", MonIDString);

   sprintf (cmdString,
      "/sbin/evlnotify -p --add \"%s\" --filter \"data contains
\\\"Monitor_ID=%s\\\"\"",
      cmdAction, MonIDString);
   //printf ("\n%s\n", cmdString);
   system( cmdString );

   /* Define the signal handler... */
   memset (&SigRTAction, 0, sizeof (struct sigaction));
   SigRTAction.sa_flags = SA_SIGINFO; // | SA_RESTART;

   /* use myListener for this Monitor's events. */
   SigRTAction.sa_handler = (void (*) (int)) myListener;

   /* ... signo_offset can typically default to 0, if
      ... you are only using a single listener. */

   if ( sigaction (SIGRTMIN + 1 + signo_offset, &SigRTAction, NULL) < 0 )
     PrintError ("Sigaction failed for new SIGRTMIN. %s", "sigaction", ErrorStr);

   /* A query string will typically consist of a monitor's GUID
      e.g. "data contains \"Monitor_ID=cc63f3bb-d74b-41a0-ab46-5d81579c9b11\"" */

   /* Register the query string and check it for basic syntax */

   if ( posix_log_query_create (newquery_string,
               POSIX_LOG_PRPS_NOTIFY,
               &newposix_query,
               error_string,
               POSIX_LOG_MEMSTR_MAXLEN) ) {
     fprintf (stderr, "ERROR: could not create newposix_query! %s.\n",
error_string);
     return -1;
   }
```

```
  /* Initialize mynotification with a function pointer
     which is to be the callback function. */
  mynotification.sigev_notify_function = (void (*) (union sigval)) myListener;

  /*  Set the desired POSIX signal value.
   *  Your application only needs one callback registration,
   *  so use SIGRTMIN+1. For additional callback functions
   *  you should increment sigev_signo to be distinct values
   *  between SIGRTMIN to SIGRTMAX (a range of 32 values).
   */

  /* sigev_signo also needs to match the sigaction number used above */
  mynotification.sigev_signo = SIGRTMIN + 1 + signo_offset;

  /* sival_int can be used by myListener() to
     identify the monitor which triggers these events */
  mynotification.sigev_value.sival_int = (int) newmonitor;
  mynotification.sigev_notify = SIGEV_SIGNAL;

  if ( posix_log_notify_add (&newposix_query,
                      &mynotification,
            0, /* -1 never tell,  0 tell all,  1 tell me once */
            &newquery_handle) ) {
          PrintError ("%s", "posix_log_notify_add", ErrorStr);
    return -1;
  }

  fflush (stdout);
  return 0;
}


int  CreateWatermarkMonitor (rmStatisticId statKey, enum rmWatermarkType type)
{
  rmMonitorControl mControl;
  rmMonitorConfiguration  mConfig;
  rmHandle    p;
  rmValue       value;
  int       i, retval;

  clearMonitorUID (mControl);
  clearDataCapture (mControl);
  mControl.monitorType      = rmWatermarking;
  mControl.location       = rmDaemon;
  mControl.monitoringRate     = 1;
  mControl.monitoringInterval    = 0;
  mControl.microMonitoringRate   = 000;

  mConfig.monitorType       = rmWatermarking;
  mConfig.statisticTransform     = rmNone;
  mConfig.statisticKey      = statKey;
  mConfig.RMThresholdType     = type;

  if ((p = rmCreateMonitor (&mConfig, &mControl)) == NULL) {
    switch (errno) {
      case EBUSY:
```

```
          PrintError ("The daemon has exceeded its CPU utilization level",
                "rmCreateMonitor");
          break;
        case ENXIO:
          PrintError ("The sybsystem, resource, or statistic id in "
                "rmMonitorConfiguration is not available", "rmCreateMonitor");
          break;
        case EEXIST:
          PrintError ("The provided monitor uid exists already", "rmCreateMonitor");
          break;
        case ENOENT:
          PrintError ("The data capture library is not available",
"rmCreateMonitor");
          break;
        case EINVAL:
          PrintError ("Any other configuation or control data is invalid",
                "rmCreateMonitor");
          break;
        default:
          PrintError ("%s\n", "rmCreateMonitor", ErrorStr);
      }
      exit (1);
    }
    if ( (retval = rmStartMonitor (p)) ) {
      if (retval == ESRCH)
        PrintError ("handle [%p] is invalid", "rmStartMonitor", p);
      else if (retval == EBUSY)
        PrintError ("the daemon process has exceeded its CPU usage limit",
"rmStartMonitor");
      else
        PrintError ("%s", "rmStartMonitor", ErrorStr);
      exit (1);
    }

    /* Print the upper bound of the resource */
    rmGetUpperBound (statKey, &value);
    printf ("Upper Bound for \"%s\" is: [%u]\n",
      GetStatDesc (statKey.SubsystemId, statKey.StatisticId), value.rmValueU32);

    /* Print the High Watermark in 10 minutes */
    for (i = 0; i < 10; i++) {

      rmMonitorState   mstat;
      rmStatisticValue state;

      /* Get the Monitor State */
      if ( (retval = rmGetMonitorState (p, &mstat)) ) {
        if ( retval == ESRCH )
          PrintError ("handle [%p] is invalid", "rmGetMonitorState", p);
        else
          PrintError ("%s", "rmStartMonitor", ErrorStr);
        exit (1);
      }
      state = mstat.lastMonitoredValue;

      if ( rmGetCurrentValue (state.Stat, &value) )
        PrintError ("%s", "rmGetCurrentValue", ErrorStr);
```

```
    printf ("\"%s\" Current Value: [%u]  High WaterMark: [%u]\n",
      GetStatDesc (state.Stat.SubsystemId, state.Stat.StatisticId),
      value.rmValueU32, mstat.typeState.watermark.highWatermark.rmValueU32);

    sleep (1);
  }
  /* Delete the monitor */
  if ( rmDeleteMonitor (p) )
    PrintError ("%s", "rmDeleteMonitor", ErrorStr);

  return 0;
}


int  CreateThresholdMonitor (rmStatisticId statKey, enum rmThresholdType type,
                enum rmThresholdTestCondition condition,
                u_int32_t value, char *cmdAction)
{
  rmMonitorControl mControl;
  rmMonitorConfiguration  mConfig;
  rmHandle    p;
  int      retval;

  clearMonitorUID (mControl);
  clearDataCapture (mControl);
  mControl.monitorType      = rmThresholding;
  mControl.location       = rmDaemon;
  mControl.monitoringRate     = 4;
  mControl.monitoringInterval    = 9;
  mControl.RMThresholdTolerance.rmValueU32= 0;
  mControl.RMThresholdSamples    = 1;
  mControl.RMThresholdLoggingRate     = 2;

  mConfig.monitorType       = rmThresholding;
  mConfig.statisticKey      = statKey;
  mConfig.statisticTransform     = rmNone;
  mConfig.RMThresholdType     = type;
  mConfig.RMThresholdValue.rmValueU32 = value;
  mConfig.RMThresholdCondition   = condition;
  mConfig.RMThresholdPrecondition     = rmNoPrecondition;
  mConfig.RMThresholdEventSeverity = ME_LOG_CRIT;
  mConfig.RMThresholdCancelEventSeverity  = ME_LOG_INFO;

  if ((p = rmCreateMonitor (&mConfig, &mControl)) == NULL) {
    switch (errno) {
      case EBUSY:
        PrintError ("The daemon has exceeded its CPU utilization level",
              "rmCreateMonitor");
        break;
      case ENXIO:
        PrintError ("The sybsystem, resource, or statistic id in"
              "rmMonitorConfiguration is not available", "rmCreateMonitor");
        break;
      case EEXIST:
        PrintError ("The provided monitor uid exists already", "rmCreateMonitor");
        break;
```

```
       case ENOENT:
         PrintError ("The data capture library is not available",
"rmCreateMonitor");
         break;
       case EINVAL:
         PrintError ("Any other configuation or control data is invalid",
               "rmCreateMonitor");
         break;
       default:
         PrintError ("%s", "rmCreateMonitor", ErrorStr);
     }
     exit (1);
   }
   registerListener (p, eventListener, cmdAction, 1);
   if ((retval = rmStartMonitor (p)) != 0) {
     if (retval == ESRCH)
       PrintError ("handle [%p] is invalid", "rmStartMonitor", p);
     else if (retval == EBUSY)
       PrintError ("the daemon process has exceeded its CPU usage limit",
"rmStartMonitor");
     else
       PrintError ("%s", "rmStartMonitor", ErrorStr);
     exit (1);
   }

   /* Delete the monitor */
   if ( rmDeleteMonitor (p) )
     PrintError ("%s", "rmDeleteMonitor", ErrorStr);

   return 0;
}


int  CreateLeakyBucketMonitor (rmStatisticId statKey, u_int32_t size, u_int32_t
value, char *cmdAction)
{
   rmMonitorControl mControl;
   rmMonitorConfiguration  mConfig;
   rmHandle    p;
   int       i, retval;

   clearMonitorUID (mControl);
   clearDataCapture (mControl);
   mControl.monitorType      = rmLeakyBucket;
   mControl.location       = rmDaemon;
   mControl.monitoringRate     = 4;
   mControl.monitoringInterval    = 9;
   mControl.RMLeakyBucketLoggingRate= 1;

   mConfig.monitorType       = rmLeakyBucket;
   mConfig.statisticKey      = statKey;
   mConfig.statisticTransform     = rmNone;
   mConfig.RMbucketSize.rmValueU64     = size;
   mConfig.RMfillValue.rmValueU32   = value;
   mConfig.RMleakyBucketEventSeverity  = ME_LOG_CRIT;

   if ((p = rmCreateMonitor (&mConfig, &mControl)) == NULL) {
```

```
    switch (errno) {
      case EBUSY:
        PrintError ("The daemon has exceeded its CPU utilization level",
              "rmCreateMonitor");
        break;
      case ENXIO:
        PrintError ("The sybsystem, resource, or statistic id in"
              "rmMonitorConfiguration is not available", "rmCreateMonitor");
        break;
      case EEXIST:
        PrintError ("The provided monitor uid exists already", "rmCreateMonitor");
        break;
      case ENOENT:
        PrintError ("The data capture library is not available",
"rmCreateMonitor");
        break;
      case EINVAL:
        PrintError ("Any other configuation or control data is invalid",
              "rmCreateMonitor");
        break;
      default:
        PrintError ("%s", "rmCreateMonitor", ErrorStr);
    }
    exit (1);
  }
  registerListener (p, eventListener, cmdAction, 1);
  if ((retval = rmStartMonitor (p)) != 0) {
    if (retval == ESRCH)
      PrintError ("handle [%p] is invalid", "rmStartMonitor", p);
    else if (retval == EBUSY)
      PrintError ("the daemon process has exceeded its CPU usage limit",
"rmStartMonitor");
    else
      PrintError ("%s", "rmStartMonitor", ErrorStr);
    exit (1);
  }

  for (i = 0; i < 10; i++) {
    PrintCurrentValue (statKey);
    sleep (1);
  }
  /* Delete the monitor */
  if ( rmDeleteMonitor (p) )
    PrintError ("%s", "rmDeleteMonitor", ErrorStr);

  return 0;
}


const rmStatisticId *GetStatKey (rmString subsystemUUID, int resourceId, int
statisticId)
{
  static rmStatisticId  statKey;

  uuid_parse (subsystemUUID, statKey.SubsystemId);
  statKey.ResourceId = resourceId;
  statKey.StatisticId  = statisticId;
```

```
  return &statKey;
}


int  main (int argc, char **argv)
{
  const rmStatisticId * statKey;

  /* Find out and print all the subsystems and their statistics */
  discover_resource ();
  discover_monitors ();

  /* Set a statistics structure to tell what you want to monitor */
  statKey = GetStatKey (RM_KERNEL_UUID_STRING, 0, KL_STAT_OPEN_FILE_DESCRIPTORS);
  PrintCurrentValue (*statKey);

  /* Create a threshold monitor */
  CreateThresholdMonitor (*statKey, rmThreshold, rmValueIsAtOrAbove, 5,
         "wall 'You can do your action here'");

  /* Create a watermark monitor */
  /* Note: Change below 2051 and 2049 to your own resource id */
  statKey = GetStatKey (FILESYSTEM_GUID, 2051,
FILESYSTEM_STAT_PERCENT_USED_SPACE_ID);
  CreateWatermarkMonitor (*statKey, rmHighWatermark);
  PrintCurrentValue (*statKey);

  /* Create a leakybucket monitor */
  /* Copying a big file to the resource 2049 in 10 minutes can generate a event log
*/
  statKey = GetStatKey (FILESYSTEM_GUID, 2049,
FILESYSTEM_STAT_PERCENT_USED_SPACE_ID);
  PrintCurrentValue (*statKey);
  CreateLeakyBucketMonitor (*statKey, 1, 0, "wall 'You can do your action here'");

  return 0;
}
```

Appendix – Sample C++

```
/***************************************************************************
                          main.cpp  -  description
                             -------------------
    begin                : Sat May 18 09:37:59 EDT 2002
    copyright            : (C) 2002 by banks
    email                : Ken.Banks@Intel.com
 ***************************************************************************/


/***************************************************************************
 *                                                                         *
 *   This program is provided with the Resource Monitor User Guide to      *
 *   server as sample code for using the Resource Monitor API's.           *
 *                                                                         *
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; either version 2 of the License, or     *
 *   (at your option) any later version.                                   *
 *                                                                         *
 ***************************************************************************/

#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include <ResourceMonitor.h>
#include <MonitoringEventSchema.h>
#include <RM_KernelMonitor.h>
#include <RM_FileSystemMonitor.h>
#include <RM_ProcessMonitor.h>
#include <TextDataCapture.h>
#include "MyStrings.h"
#include <errno.h>
#include <getopt.h>
#include <iostream.h>
#include <linux/stddef.h>
#include <netdb.h>
#include <paths.h>
#include <posix_evlog.h>
#include <signal.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/file.h>
#include <sys/signal.h>
#include <sys/uio.h>
#include <sys/wait.h>
#include <time.h>
```

```
#include <unistd.h>
#include <wchar.h>
#define PAUSE cout << "Press ENTER to continue..." << endl;cin.get();

void discovery();
void create_monitors();
void view_and_modify_monitors();

bool createWatermark( rmStatisticId StatKey,
                      rmWatermarkType WMtype,
                      RMuid dataCaptureID);

bool createThreshold( rmStatisticId            StatKey,
                      rmThresholdType           Ttype,
                      rmThresholdTestCondition Condition,
                      char                    *ThresholdValue,
                      char                    *cmdAction,
                      RMuid                    dataCaptureID);

bool createLeakyBucket( rmStatisticId    StatKey,
                        rmTimeInterval   bucketTimer,
                        char            *bucketSize,
                        char            *fillRate,
                        char            *cmdAction,
                        RMuid            dataCaptureID);

void printCurrentValue( rmStatisticId StatKey);
void eventListener( int signo, siginfo_t  *info, void *ignored);
void myEventResponse( const char *event_string);
bool getEventRecord( posix_log_recid_t recid, char *event_string);
bool registerListener( ResourceMonitor::RMmonitor *newmonitor,
                       void (*myListener) (int, siginfo_t *, void *),
                       char cmdAction[],
                       int signo_offset);
int PrintStatisticInfo( rmStatisticInfo *StatInfo);
int PrintSubSysInfo( rmUID SubsystemId);
int PrintResourceInfo( rmUID mySubsystemId, rmID myResId);
int PrintResourceInfo( rmResourceInfo *ResInfo);
int PrintMonitorState( ResourceMonitor::RMmonitor *p);
int PrintMonitorState( rmMonitorState  state);
int PrintMonitorConfigControl( ResourceMonitor::RMmonitor *p);
int PrintMonitorControl( rmMonitorControl moncontrol);
int PrintMonitorConfig( rmMonitorConfiguration monconfig);
int getStatSize( rmStatisticKey statKey);
void cleanup( int signum);

int main(int argc, char *argv[])
{

   cout << "This program is provided as sample code \n"
        << "for using the Resource Monitor API's.\n\n"
        << "WARNING:  This program creates SAMPLE monitors\n"
        << "          on this server.\n\n"
        << "Press CTRL-C to ABORT.\n";
   PAUSE;
```

```
   printf( "Signals will be masked so that only CTRL-C can exit.\n");
   PAUSE;
   struct sigaction sa_old;
   struct sigaction sa_new;
   sa_new.sa_handler = cleanup;
   sigemptyset( &sa_new.sa_mask );
   sigaddset( &sa_new.sa_mask, SIGINT);
   sigaddset( &sa_new.sa_mask, SIGSEGV);
   sigaddset( &sa_new.sa_mask, SIGHUP);
   sigaddset( &sa_new.sa_mask, SIGTERM);
   sa_new.sa_flags = 0;
   if(  sigaction( SIGINT, &sa_new, &sa_old) == -1) {
       perror( " ERROR: Failed sigaction().\n");
       return EXIT_FAILURE;
   }


   printf( "\nDiscover subsystems, resources and statistics...\n");
   PAUSE;
   discovery();

   printf( "\nCreate SAMPLE monitors...\n");
   PAUSE;
   create_monitors();

   printf( "\nView config/control info for all monitors.\n");
   PAUSE;
   view_and_modify_monitors();


   printf( "\nException events will be printed to stdout...\n");
   cout << "Press CTRL-C to EXIT.\n";
   while( true){

      sleep( 1 );
   }

   return EXIT_SUCCESS;
}

void discovery()
{

  size_t nSub = ResourceMonitor::rmGetSubsystemCount();
  size_t nRes;  // <-- entity counts
  size_t nStat;

  rmSubsystemInfo *SubBuffer = (rmSubsystemInfo *)
                               malloc( sizeof(rmSubsystemInfo) * nSub);
  rmSubsystemInfo *Subp = SubBuffer;    // <--|
  rmResourceInfo  *ResBuffer,  *Resp;   // <--| buffers and iterators
  rmStatisticInfo *StatBuffer, *Statp;  // <--|
  size_t si, ri, sti;

  int result=0;

  errno=0;
```

```
  result=ResourceMonitor::rmGetAvailableSubsystems( Subp, nSub );
  if (errno==EAGAIN ) {
      fprintf( stderr, "Subsystem buffer %d is too small!\n", nSub);
      perror("rmGetAvailableSubsystems() error=%d %s\n");
  }

  for( si=1 ; si<=nSub; si++, Subp++ ) {

     PrintSubSysInfo( Subp->id);
     nRes = Subp->noResources;
     ResBuffer = (rmResourceInfo *) malloc( sizeof(rmResourceInfo ) * nRes );
     Resp = ResBuffer;
     errno=0;
     ResourceMonitor::rmGetActiveResources( Subp->id, Resp, nRes);
     if (errno) { //==EAGAIN ) {
         printf("rmGetActiveResources() error=%d %s\n", errno, strerror( errno));
         if (errno==EAGAIN) cout << "Resource size: " << nRes << " is too small!\n";
     }
     if( nRes != Subp->noResources ) {
        printf( "inconsistent no. of stats: %d %d\n", nStat, Subp->noStatistics);
     }
     for( ri=0 ; ri<nRes; ri++, Resp++ ) {
               PrintResourceInfo( Resp);
     }
     free( ResBuffer);

     nStat = Subp->noStatistics;
     if(  nStat>0 ) {
          StatBuffer = (rmStatisticInfo *) malloc( sizeof(rmStatisticInfo ) *
nStat);
          Statp = StatBuffer;

          ResourceMonitor::rmGetAvailableStatistics( Subp->id, Statp, nStat);

          if( nStat != Subp->noStatistics )
             printf( "inconsistent no. of stats: %d %d\n", nStat, Subp-
>noStatistics);

          for( sti=0 ; sti<nStat ; sti++, Statp++ ) {
                  PrintStatisticInfo( Statp);
          }
          fflush( stdout);
          free( StatBuffer);
      }
  }
  free( SubBuffer);

}    // discovery()


void create_monitors()
{
  RMuid  SubID;
  RMuid  dataCaptureID;
  rmStatisticId StatKey;

   dataCaptureID.clear();
```

```
   /*  Insert either of the lines below to enable
    *  DATA CAPTURE for any or all monitor examples...
    *
    *  dataCaptureID.parse( TEXT_TABLE_DATACAPTURE_UUID_STRING);
    *  dataCaptureID.parse( TEXT_XML_DATACAPTURE_UUID_STRING);
    */

/*
 *   WATERMARK EXAMPLE
 */
// This watermark will monitor OPEN FILE DESCRIPTORS on "/"
   SubID.parse( RM_KERNEL_UUID_STRING );
   StatKey.SubsystemId = SubID.id;
   StatKey.ResourceId  = 0;
   StatKey.StatisticId = KL_STAT_OPEN_FILE_DESCRIPTORS;

   printCurrentValue( StatKey );

   createWatermark( StatKey, rmHighWatermark, dataCaptureID);

// This watermark can monitor VIRTUAL MEMORY for a process
   SubID.parse( RM_PROCESS_UUID_STRING );
   StatKey.SubsystemId = SubID.id;
   StatKey.ResourceId  = 1; // <-- ResourceId is process id 1 (init)
   StatKey.StatisticId = STAT_vsize_ID;

   createWatermark( StatKey, rmHighWatermark, dataCaptureID);

/*
 *   THRESHOLD EXAMPLE
 */
   SubID.parse( FILESYSTEM_GUID );
   StatKey.SubsystemId = SubID.id;
   StatKey.ResourceId  = 2050; // <-- Your ResourceId may vary
   StatKey.StatisticId = FILESYSTEM_STAT_PERCENT_USED_SPACE_ID;

   createThreshold( StatKey,
                    rmThreshold,
                    rmValueIsAtOrAbove,
                    "95",
                    "PercentUsedSpace_Response.sh",
                    dataCaptureID);
/*
 *   LEAKYBUCKET EXAMPLE
 */
#define SCSI_GUID "cad21c0e-53eb-4261-8e2d-ee8aa5a49b09"
#define SCSI_DISK_ERRORS 3
   SubID.parse( SCSI_GUID );
   StatKey.SubsystemId = SubID.id;
   StatKey.ResourceId  = 0; // <-- Your ResourceId may vary
   StatKey.StatisticId = SCSI_DISK_ERRORS;

   createLeakyBucket( StatKey,
                      86400,   // 24 hour period
                      "5",     // bucket - intial allowable errors
                      "4",     // fill - allowable error count
                      "DiskErrors_Response.sh",
```

```
                    dataCaptureID);

}

void view_and_modify_monitors()
{
    ResourceMonitor::RMmonitor *newp;
    size_t Mcount = ResourceMonitor::rmGetMonitorCount();

    if( Mcount ) {
       rmMonitorInfo *MonBuffer = (rmMonitorInfo *)malloc(sizeof( rmMonitorInfo ) *
Mcount);
       rmMonitorInfo *pMon = MonBuffer;
       printf( "Discovered ( %d ) monitors with rmGetMonitors.\n", Mcount);
       if( ResourceMonitor::rmGetMonitors( pMon, &Mcount ) != 0 ) {
             perror("view_and_modify_monitors(): rmGetMonitors()");
             free(MonBuffer);
             return;
       }
       for( size_t p=0; p < Mcount ; p++ ) {
           try {
                /*
                 *   This "new" does NOT create a new monitor object.
                 *   It simply returns a class pointer to an existing
                 *   monitor object identified with this "uid".
                 */
               newp= new ResourceMonitor::RMmonitor( pMon[p].uid);
               cout << newp->GetMonitorDescription( rmShortDescription ) << " : ";
               PrintMonitorState( newp );
               PrintMonitorConfigControl( newp );

           }
           catch(...) {
                    perror("Monitor not found: ");
                    RMuid toPrint( pMon[p].uid );
                    toPrint.print(cout);
                    cout << "\n";
           }

       }
       free( MonBuffer);
    }
    fflush( stdout);
    return;

}

bool  createWatermark( rmStatisticId StatKey,
                       rmWatermarkType WMtype,
                       RMuid dataCaptureID)
{
  rmMonitorControl      moncontrol;
      clearMonitorUID(  moncontrol );
      moncontrol.monitorType         = rmWatermarking;
      moncontrol.location            = rmDaemon;
      moncontrol.monitoringInterval  = 0;   // 0 seconds means forever
      moncontrol.monitoringRate      = 5;   // seconds
```

```
      moncontrol.microMonitoringRate = 000; // micro-seconds
      moncontrol.dataCapture         = dataCaptureID.id;

  rmMonitorConfiguration monconfig;
      monconfig.statisticKey    = StatKey;
      monconfig.monitorType     = rmWatermarking;
      monconfig.RMWatermarkType = WMtype;

  ResourceMonitor::RMmonitor *p;
  try { p= new ResourceMonitor::RMmonitor(  monconfig, &moncontrol);
      }
  catch(...) {
               perror("createWatermark FAILED!");
               return false;
  }

  //  TODO - More descriptive name should be set here...
  string NewName = "SAMPLE Watermark for \"" +
                    (string) ResourceMonitor::rmGetStatisticDescription(
                                    StatKey.SubsystemId,
                                    StatKey.StatisticId,
                                    rmShortDescription) + "\"";
  cout << "Create " << NewName << "\n";
  p->SetMonitorDescription( rmShortDescription,
                            (rmString) NewName.c_str());
  p->StartMonitor( );
  return true;

}   // createWatermark()

bool createThreshold(  rmStatisticId            StatKey,
                       rmThresholdType           Ttype,
                       rmThresholdTestCondition Condition,
                       char                    *ThresholdValue,
                       char                    *cmdAction,
                       RMuid                    dataCaptureID)
{
   rmMonitorControl     moncontrol;
      clearMonitorUID(  moncontrol );
      moncontrol.monitorType                    = rmThresholding;
      moncontrol.location                       = rmDaemon;
      moncontrol.monitoringInterval             = 0;   // 0 seconds means forever
      moncontrol.monitoringRate                 = 5;   // seconds
      moncontrol.microMonitoringRate            = 000; // micro seconds
      moncontrol.RMThresholdTolerance.rmValueU32 = 0;
      moncontrol.RMThresholdSamples             = 1;
      moncontrol.RMThresholdLoggingRate         = 2;   // seconds
      moncontrol.dataCapture                    = dataCaptureID.id;

   rmMonitorConfiguration monconfig;
   char *pEnd;
      switch( getStatSize( StatKey )) {
         case rmSizeS32:
                        monconfig.RMThresholdValue.rmValueS32 = atoi(
ThresholdValue );
                        break;
         case rmSizeU32:
```

```
                        monconfig.RMThresholdValue.rmValueU32 = strtoul(
ThresholdValue, &pEnd, 0);
                        break;
         case rmSizeS64:
                        monconfig.RMThresholdValue.rmValueS64 = strtoll(
ThresholdValue, &pEnd, 0);
                        break;
         case rmSizeU64:
                        monconfig.RMThresholdValue.rmValueU64 = strtoull(
ThresholdValue, &pEnd, 0);
                        break;
         default:
                        fprintf( stderr, "Invalid statistic size!\n");
                        return false;
                        break;
      }
      monconfig.statisticKey                  = StatKey;
      monconfig.monitorType                   = rmThresholding;
      monconfig.RMThresholdType               = Ttype;
      monconfig.RMThresholdTransform          = rmNone;
      monconfig.RMThresholdCondition          = Condition;
      monconfig.RMThresholdPrecondition       = rmNoPrecondition;
      monconfig.RMThresholdEventSeverity      = ME_LOG_CRIT;
      // Cancel Severity is only used for rmBiDirectionalThreshold
      monconfig.RMThresholdCancelEventSeverity = ME_LOG_INFO;

  ResourceMonitor::RMmonitor *p;
  try { p= new ResourceMonitor::RMmonitor(  monconfig, &moncontrol);
      }
  catch(...) {
               perror("createThreshold FAILED!\n");
               return false;
  }

  registerListener( p, eventListener, cmdAction, 1);

  //  TODO - More descriptive name should be set here...
  string NewName= "SAMPLE Threshold for \"" +
                   (string) ResourceMonitor::rmGetStatisticDescription(
                                    StatKey.SubsystemId,
                                    StatKey.StatisticId,
                                    rmShortDescription) + "\"";
  cout << "Create " << NewName << "\n";
  p->SetMonitorDescription( rmShortDescription,
                            (rmString) NewName.c_str());
  p->StartMonitor( );
  return true;

} // createThreshold()


bool createLeakyBucket( rmStatisticId    StatKey,
                        rmTimeInterval   bucketTimer,
                        char            *bucketSize,
                        char            *fillRate,
                        char            *cmdAction,
                        RMuid            dataCaptureID)
```

```
{
  rmMonitorControl moncontrol;
      clearMonitorUID( moncontrol );
      moncontrol.monitorType              = rmLeakyBucket;
      moncontrol.location                 = rmDaemon;
      moncontrol.monitoringInterval       = 0; // 0 seconds means forever
      moncontrol.monitoringRate           = bucketTimer;   // seconds
      moncontrol.microMonitoringRate      = 000;     // micro seconds
      moncontrol.RMLeakyBucketLoggingRate  = 2;
      moncontrol.dataCapture              = dataCaptureID.id;

   rmMonitorConfiguration monconfig;
   char *pEnd;
      monconfig.statisticKey              = StatKey;
      monconfig.monitorType               = rmLeakyBucket;
      monconfig.RMleakyBucketEventSeverity = ME_LOG_CRIT;
      switch( getStatSize( StatKey )) {
         case rmSizeS32:
                         monconfig.RMbucketSize.rmValueS32    = atoi( bucketSize );
                         monconfig.RMfillValue.rmValueS32     = atoi( fillRate );
                         break;
         case rmSizeU32:
                         monconfig.RMbucketSize.rmValueU32    = strtoul( bucketSize,
&pEnd, 0);
                         monconfig.RMfillValue.rmValueU32     = strtoul( fillRate,
&pEnd, 0);
                         break;
         case rmSizeS64:
                         monconfig.RMbucketSize.rmValueS64     = strtoll(
bucketSize, &pEnd, 0);
                         monconfig.RMfillValue.rmValueS64      = strtoll( fillRate,
&pEnd, 0);
                         break;
         case rmSizeU64:
                         monconfig.RMbucketSize.rmValueU64     = strtoull(
bucketSize, &pEnd, 0);
                         monconfig.RMfillValue.rmValueU64      = strtoull( fillRate,
&pEnd, 0);
                         break;
         default:
                         fprintf( stderr, "Invalid statistic size!\n");
                         return false;
                         break;
      }

  ResourceMonitor::RMmonitor *p;
  try { p= new ResourceMonitor::RMmonitor(  monconfig, &moncontrol);
      }
  catch(...) {
              perror("createLeakyBucket FAILED!\n");
              return false;
  }

  registerListener( p, eventListener, cmdAction, 1);

  //  TODO - More descriptive name should be set here...
  string NewName= "SAMPLE LeakyBucket for \"" +
```

```
                    (string) ResourceMonitor::rmGetStatisticDescription(
                                    StatKey.SubsystemId,
                                    StatKey.StatisticId,
                                    rmShortDescription) + "\"";
  cout << "Create " << NewName << "\n";
  p->SetMonitorDescription( rmShortDescription,
                           (rmString) NewName.c_str());
  p->StartMonitor( );
  return true;

} //  createLeakyBucket()



void printCurrentValue( rmStatisticId StatKey)
{
      rmValue currValue;
      if( (ResourceMonitor::rmGetCurrentValue(StatKey, &currValue)) !=0) {
           perror("ResourceMonitor::rmGetCurrentValue");
      } else {
           printf("\nCurrent value for %s: %u\n",
                   ResourceMonitor::rmGetStatisticDescription(
                        StatKey.SubsystemId,
                        StatKey.StatisticId,
                        rmShortDescription),
                   currValue.rmValueU32);
      }
}


/*  A prior call to registerListener() uses posix_log_notify_add()
 *  to create a query and register this function as the callback.
 *  If this query matches any incoming event record, then this
 *  eventListener() function will be called.
 *
 *  This function responds to events related to
 *  Threshold and LeakyBuckets exceptions.
 */
void  eventListener( int         signo,
                     siginfo_t  *info,
                     void       *ignored)
{
  posix_log_recid_t recid;
  char incoming_event_msg_string[ POSIX_LOG_ENTRY_MAXLEN+128];
  sigset_t old_sig;
  sigset_t nold;

  sigfillset(&nold);
  sigdelset(&nold, SIGINT);
  sigdelset(&nold, SIGQUIT);
  sigprocmask(SIG_SETMASK, &nold, &old_sig);

  /* Get the recid of the event that caused this function call. */
  posix_log_siginfo_getrecid(info, ignored, &recid);

  if( getEventRecord( recid, incoming_event_msg_string) ) {
        /* TODO - a fork() is more appropriate here, because
```

```
           we should not waste time inside the eventListener() */
        myEventResponse( incoming_event_msg_string);
  } else {
        cout << "\nCan't find offending RECID: " << recid << "\n";
  }

  fflush(stdout);

  sigprocmask(SIG_SETMASK, &old_sig, NULL);
  return;
}  // eventListener()

void myEventResponse( const char *event_string)
{
   cout << "\n" << event_string << "\n";

  /* TODO - parse the event_string here for key
   *        information to act upon, such as event_type,
   *        severity, Subsystem_Name, Resource_Name
   *        Statistic_Name, Observed_Value, Threshold_Value,
   *        etc...
   *        (more fields found in MonitoringEventSchema.h)
   *
   * NOTE:  An event_type of ME_EVENT_CANCELED denotes that
   *        a bidirectional threshold is no longer exceeded,
   *        i.e. the statistic has returned to normal range.
   */

   fflush(stdout);
}

bool getEventRecord( posix_log_recid_t recid,
                     char *event_string)
{
  char   recid_query_string[POSIX_LOG_MEMSTR_MAXLEN];
  char   error_string[ POSIX_LOG_MEMSTR_MAXLEN];
  posix_log_query_t  recid_posix_query;
  posix_logd_t       logdes;
  struct posix_log_entry myentry, *entry;
  int ret;

  strcpy( event_string, "No Event String Yet");

  // Searching the event log for record where id= recid
  sprintf( recid_query_string, "recid = %d", recid);
  if(posix_log_query_create( recid_query_string, POSIX_LOG_PRPS_SEEK,
                             &recid_posix_query, error_string,
POSIX_LOG_MEMSTR_MAXLEN) != 0) {
     fprintf( stderr, "ERROR: Could not create query for recid: %d  %s.\n",
                      recid, error_string);
     exit(1);
  }

  /* Let's get the entire event log entry */
  entry=&myentry;
  if(posix_log_open(&logdes, NULL)==0) {
```

```
       // Just read from bottom and get mesg. This is to be used
       // with automated tests or faster response demos
       // Register the query string and check syntax
       errno=0;
       if((ret=posix_log_seek(logdes,NULL,POSIX_LOG_SEEK_END))==0) {
          if((ret=posix_log_seek(logdes, &recid_posix_query,
POSIX_LOG_SEEK_BACKWARD))==0) {
             if((ret=posix_log_read( logdes, entry,
                                    event_string, POSIX_LOG_ENTRY_MAXLEN-1))==0) {
                   printf( "Found RECID: %d MESG: %s\n",
                            entry->log_recid, event_string);
                   return true;
             } else {
                      printf( "\n ERROR: (READ) posix_log_read FAILED to find
recid=%d  err=%d errno=%d.  Attempt to process this event record: \"%s\"\n",
                               recid, (unsigned int) ret, errno, event_string);
                      return false;
             }
          } else {
              sleep( 1);
              errno=0;
              if((ret=posix_log_seek(logdes,NULL,POSIX_LOG_SEEK_END))==0) {
                 if((ret=posix_log_seek(logdes, &recid_posix_query,
POSIX_LOG_SEEK_BACKWARD))==0) {
                   if((ret=posix_log_read( logdes, entry,
                                           event_string, POSIX_LOG_ENTRY_MAXLEN-
1))==0) {
                              printf( "SUCCESS: FIRST backward seek FAILED! SECOND seek
found RECID: %d  MESG: %s\n",
                                       entry->log_recid, event_string);
                   } else {
                              printf( "\n ERROR: First backward seek FAILED!  SECOND
seek (READ) posix_log_read FAILED to find recid=%d  err=%d errno=%d.  Attempt to
process this event record: \"%s\"\n",
                                        recid, (unsigned int) ret, errno,
event_string);
                              return false;
                   }

                  } else {
                    printf( "\n ERROR: First and SECOND backward seeks FAILED!
(SEEK) posix_log_seek FAILED to find recid=%d  err=%d errno=%d.  Attempt to process
this event record: \"%s\"\n",
                             recid, ret, errno, event_string);
                    return false;
                  }
              } else {
                       fprintf(stderr, "ERROR: FAILED Could not seek AGAIN to end of
log file. ret=%d  errno=%d\n", ret, errno);
                       return false;
              }
          }
          if( posix_log_query_destroy( &recid_posix_query)   !=   0)   {
              printf( "WARNING: posix_log_query_destroy() failed for
recid_posix_query. ");
          }
```

```
       } else {
                fprintf(stderr, "ERROR: Could not seek to end of log file. ret=%d
errno=%d\n", ret, errno);
                return false;
       }
    posix_log_close(logdes);
  } else {
          printf("\nERROR: could not open the event log file. errno=%d.\n", errno);
          return false;
  }
  fflush(stdout);
  return true;
} // getEventRecord( )

bool registerListener( ResourceMonitor::RMmonitor *newmonitor,
                       void (*myListener) (int, siginfo_t *, void *),
                       char cmdAction[],
                       int signo_offset)
{
  struct sigevent mynotification;      //  Remember the monitor's query_handle
  posix_log_notify_t newquery_handle;  //  and its query so we can stuff them
  posix_log_query_t  newposix_query;   //  into the Monitorset to use later.
  char   error_string[ POSIX_LOG_MEMSTR_MAXLEN];
  rmMonitorInfo mInfo;
  char newquery_string[POSIX_LOG_MEMSTR_MAXLEN];

   if( newmonitor->GetMonitorInfo( &mInfo) != 0 ) {
       perror("rmGetMonitorInfo");
       return false;
   }
   char  MonIDString[ MAX_GUID_STRLEN ];
   RMuid MonID( mInfo.uid );
   MonID.unparse( &MonIDString[0] );
   sprintf( newquery_string, "data contains \"Monitor_ID=%s\"", MonIDString);

   char cmdString[POSIX_LOG_MEMSTR_MAXLEN];
   sprintf( cmdString,
   "/sbin/evlnotify -p --add \"%s\" --filter \"data contains
\\\"Monitor_ID=%s\\\"\"",
                      cmdAction,                                  MonIDString);
   cout << "\n" << cmdString << "\n";
   system( cmdString );

   //  Define the signal handler...
   static struct sigaction SigRTAction;
   (void) memset(&SigRTAction, 0, sizeof(struct sigaction));
    SigRTAction.sa_flags = SA_SIGINFO; // | SA_RESTART;

   //  use myListener for this Monitor's events.
    SigRTAction.sa_handler = (void (*) (int)) myListener;

   //  ... signo_offset can typically default to 0, if
   //  ... you are only using a single listener.

   if (sigaction( SIGRTMIN + 1 + signo_offset,
                  &SigRTAction, NULL) < 0){
       fprintf(stderr, " sigaction failed for new SIGRTMIN.\n");
```

```
        perror("sigaction");
    }

    // A query string will typically consist of a monitor's GUID
    // e.g. "data contains \"Monitor_ID=cc63f3bb-d74b-41a0-ab46-5d81579c9b11\""

    // Register the query string and check it for basic syntax
    //
    if(posix_log_query_create( newquery_string,
                               POSIX_LOG_PRPS_NOTIFY,
                               &newposix_query,
                               error_string,
                               POSIX_LOG_MEMSTR_MAXLEN) != 0) {
       fprintf( stderr, "ERROR: could not create newposix_query! %s.\n",
                         error_string);
       return false;
    }

    //  Initialize mynotification with a function pointer
    //  which is to be the callback function.
    mynotification.sigev_notify_function = (void (*) (sigval)) myListener;

    /*  Set the desired POSIX signal value.
     *  Your application only needs one callback registration,
     *  so use SIGRTMIN+1. For additional callback functions
     *  you should increment sigev_signo to be distinct values
     *  between SIGRTMIN to SIGRTMAX (a range of 32 values).
     */
     // sigev_signo also needs to match the sigaction number used above
    mynotification.sigev_signo=SIGRTMIN + 1 + signo_offset;

    /*  sival_int can be used by myListener() to
        identify the monitor which triggers these events      */
    mynotification.sigev_value.sival_int = (int) newmonitor;
    mynotification.sigev_notify=SIGEV_SIGNAL;

    if(posix_log_notify_add( &newposix_query,
                             &mynotification,
                              0,  // -1 never tell,  0 tell all,  1 tell me once
                             &newquery_handle)!=0) {
            perror( "ERROR: posix_log_notify_add() failed.\n");
            return false;
    }
    fflush(stdout);
    return true;
}


int PrintStatisticInfo( rmStatisticInfo *StatInfo)
{
  printf("            [Statistic %3d]: %-30s ",
          StatInfo->id,
          ResourceMonitor::rmGetStatisticDescription(
                                    StatInfo->SubsystemId,
                                    StatInfo->id,
                                    rmShortDescription));
  printf( " (%s  %-15s  %-12s %s)\n",
```

```
          StatSizeStr[ StatInfo->size],
          StatScaleStr[StatInfo->scale],
          ResourceMonitor::rmGetStatisticDescription(
                                  StatInfo->SubsystemId,
                                  StatInfo->id,
                                  rmUnitsDescription),
          StatTypeStr[ StatInfo->type]);
  return 0;
}

int PrintSubSysInfo( rmUID SubsystemId)
{
  rmSubsystemInfo *SubInfo = (rmSubsystemInfo *) malloc( sizeof( rmSubsystemInfo ));
  errno=0;
  ResourceMonitor::rmGetSubsystemInfo( SubsystemId, SubInfo);
  if ( errno  ) {
       printf( "rmGetSubsystemInfo() error=%d %s\n", errno, strerror( errno));
       free( SubInfo);
       return -1;
  }

  printf("\n[%s] ",
         ResourceMonitor::rmGetSubsystemDescription( SubsystemId,
                                                     rmShortDescription));
  RMuid uid(SubInfo->id);
  uid.print(cout);
  cout << "\n";
  cout << SubInfo->noResources  << " Resource(s)    ";
  cout << SubInfo->noStatistics << " Statistic(s)    ";
  cout << ResourceMonitor::rmGetMonitorCount() << " Current Monitor(s)\n";
  cout << "Description: ";
  printf("%s\n",
         ResourceMonitor::rmGetSubsystemDescription( SubsystemId,
                                                     rmLongDescription));
  free( SubInfo);
  return 0;
}

int PrintResourceInfo( rmUID mySubsystemId, rmID myResId)
{
  int result;
  rmResourceInfo *ResInfo = (rmResourceInfo *)
                            malloc( sizeof(rmResourceInfo));
  errno=0;
  ResourceMonitor::rmGetResourceInfo( mySubsystemId,
                                      myResId,
                                      ResInfo);
  if ( errno  ) {
       printf( "rmGetResourceInfo() error=%d %s\n",
               errno, strerror( errno));
       free(  ResInfo);
       return -1;
  }
  result = PrintResourceInfo( ResInfo);
  free(  ResInfo);
  return result;
}
```

```
int PrintResourceInfo( rmResourceInfo *ResInfo)
{
  printf("       [Resource %u]: %s\n",
         ResInfo->id,
         ResourceMonitor::rmGetResourceDescription(
                          ResInfo->SubsystemId,
                          ResInfo->id,
                          rmShortDescription));
  return 0;
}

int PrintMonitorState( ResourceMonitor::RMmonitor *p)
{
   int result;

   rmMonitorInfo     mInfo;
   if( (result= p->GetMonitorInfo( &mInfo)) != 0 ) {
       perror("rmGetMonitorInfo ");
       return -1;
   }

   rmMonitorControl  cntrl;
   if( (result= p->GetMonitorControl( &cntrl)) != 0 ) {
       perror("rmGetMonitorControl ");
       return -1;
   }
   rmMonitorConfiguration config;
   if( (result= p->GetMonitorConfiguration( &config)) != 0 ) {
       perror("rmGetMonitorConfiguration ");
       return -1;
   }

   printf("%s ", LocationStr[cntrl.location]);

   rmStatisticInfo StatInfo;
   if( (result= ResourceMonitor::rmGetStatisticInfo(
config.statisticKey.SubsystemId,
                                    config.statisticKey.StatisticId,
                                    &StatInfo)) ) {
       perror( "GetStatisticInfo:");
       printf("\nresult=%d\n", result);
       printf( " SubsystemId=");
       RMuid uid(config.statisticKey.SubsystemId);
       uid.print(cout);
       printf( "   StatisticId=%u", config.statisticKey.StatisticId);
       cout << "\n";
       return -1;
   }

   printf("[%d] \"", StatInfo.id);

   cout << ResourceMonitor::rmGetStatisticDescription(
                                    StatInfo.SubsystemId,
                                    StatInfo.id,
                                    rmShortDescription);
   printf( "\" ");
```

```
   cout << ResourceMonitor::rmGetStatisticDescription(
                                    StatInfo.SubsystemId,
                                    StatInfo.id,
                                    rmUnitsDescription);
   printf( " ");
   cout << StatTypeStr[StatInfo.type];
   printf( " ");
   if( config.monitorType==rmThresholding) {
       printf(" %10s ", &(ThreshCondStr[ config.RMThresholdCondition ])[9]);
       printf(" %u ", config.RMThresholdValue.rmValueU32 );
   } else if( config.monitorType==rmLeakyBucket) {
       printf(" Bucket: %u ", config.RMbucketSize.rmValueU32 );
       printf(" Fill: %u ", config.RMfillValue.rmValueU32 );
   } else {
       printf(" %s ", &(WMTypeStr[config.RMWatermarkType])[2]);
   }

   printf( "\n");

   rmMonitorState    state;
   if( p->GetMonitorState( &state) != 0 ) {
       perror("GetMonitorState ");
       return -1;
   }
   return PrintMonitorState( state);
}

int PrintMonitorState( rmMonitorState  state)
{
      printf(" %s", &(MonStatusStr[state.currentStatus])[2]);
      printf(" ");
      printf("%s", (state.notificationPaused==1)? "(PAUSED)" : "");
      if( state.monitorType > 0 && state.monitorType < 4)
          printf(" %s ", &(MonTypeStr[ state.monitorType ])[2]);
      else {
          printf(" INVALID_TYPE(%d) ", state.monitorType);
      }
      printf("LastVal: %u ", state.lastMonitoredValue.Value.rmValueU32);

      string starttime(ctime( &(state.startTime)));
      string stoptime(ctime( &(state.stopTime)));
      cout << "START(" << starttime.substr(11,8) << ")    STOP("
           << stoptime.substr(11,8) << ")    ";
      switch (state.monitorType) {
           case  rmThresholding:
               {
               string lasttime(ctime( &(state.RMThresholdLastEventTime)));
               cout << " Last(" << lasttime.substr(11,8) << ") ";
               }
               break;
           case  rmWatermarking:
               {
               string lowtime(ctime( &(state.RMWatermarkLowTime)));
               string hightime(ctime( &(state.RMWatermarkHighTime)));
               cout << "(" << lowtime.substr(11,8) << ") ";
               printf(" LOW: %d    ",
                      state.RMWatermarkLow.rmValueU32);
```

```
                cout << "(" << hightime.substr(11,8) << ") ";
                printf(" HIGH: %d ",
                       state.RMWatermarkHigh.rmValueU32);
                }
                break;
            case  rmLeakyBucket:
                {
                string lasttime(ctime( &(state.RMLeakyBucketLastEventTime)));
                cout << " Last(" << lasttime.substr(11,8) << ") ";
                cout << " Level(" << state.RMLeakyBucketLevel.rmValueU32 << ") ";
                }
                break;
            default:
                cout << "UNKNOWN monitor type.";
                return -1;
                break;
        }
  return 0;
}


int PrintMonitorConfigControl( ResourceMonitor::RMmonitor *p)
{
  int result;
  cout << "       MONITOR (" << p << ")\n";

  rmMonitorConfiguration cnfig;
  p->GetMonitorConfiguration(&cnfig);
  PrintMonitorConfig( cnfig);

  rmMonitorControl cntrl;
  errno =0 ;
  result = p->GetMonitorControl(&cntrl);
  if( errno == 0 )
     PrintMonitorControl( cntrl);
  else {
     perror("GetMonitorControl:");
     printf("\nNo Control for this monitor.\n");
  }

  return 0;
}



int PrintMonitorControl( rmMonitorControl moncontrol)
{
      printf( "Monitor CONTROL:\n");
      printf( "           Type: %d (%s)\n", moncontrol.monitorType,
                                            MonTypeStr[ moncontrol.monitorType ]);
      printf( "           Muid: ");
      RMuid muid( moncontrol.uid);
      muid.print(cout);
      printf( "\n");
      printf( "       Location: %d (%s)\n", moncontrol.location,
                                            LocationStr[ moncontrol.location ]);
      printf( "       Interval: %ld\n",      moncontrol.monitoringInterval);
```

```
      printf( "           Rate: %ld\n",      moncontrol.monitoringRate);
      printf( "      microRate: %ld\n",      moncontrol.microMonitoringRate);
      if ( moncontrol.monitorType == rmThresholding ) {
           printf( "      Tolerance: %d",
moncontrol.RMThresholdTolerance.rmValueU32);
           printf( "        Samples: %d",   moncontrol.RMThresholdSamples);
           printf( "    loggingRate: %ld\n", moncontrol.RMThresholdLoggingRate);
      } else if ( moncontrol.monitorType == rmLeakyBucket ) {
           printf( "    loggingRate: %ld\n", moncontrol.RMLeakyBucketLoggingRate);
      }

      return 0;
}

int PrintMonitorConfig( rmMonitorConfiguration monconfig)
{
      printf( "Monitor CONFIGURATION:\n");
      printf( "               statID: ");
      RMuid SSuid( monconfig.statisticKey.SubsystemId);
      SSuid.print(cout);
      printf( "  R: %d  S: %d \n",
                monconfig.statisticKey.ResourceId,
                monconfig.statisticKey.StatisticId);
      printf( "                Type: %d (%s)\n",
               monconfig.monitorType,
               MonTypeStr[ monconfig.monitorType ] );

      if ( monconfig.monitorType == rmWatermarking) {

           printf( "       WatermarkType: %d (%s)\n",
                           monconfig.typeConfiguration.watermark.type,
                           WMTypeStr[ monconfig.typeConfiguration.watermark.type]);

      } else if ( monconfig.monitorType == rmThresholding ) {

           printf( "        ThresholdType: %d (%s)\n",
                          monconfig.typeConfiguration.threshold.type,
                          ThreshTypeStr[ monconfig.typeConfiguration.threshold.type
]);

           printf( "            Condition: %d (%s)\n",
                             monconfig.typeConfiguration.threshold.condition,
                             ThreshCondStr[
monconfig.typeConfiguration.threshold.condition ]);

           printf( "         Precondition: %d (%s)\n",
                            monconfig.typeConfiguration.threshold.precondition,
                            ThreshCondStr[
monconfig.typeConfiguration.threshold.precondition ] );


           printf( "        EventSeverity: %d (%s)\n",
                           monconfig.RMThresholdEventSeverity,
                           SeverityTypeStr[ monconfig.RMThresholdEventSeverity ]);

           printf( "  CancelEventSeverity: %d (%s)\n",
                     monconfig.RMThresholdEventSeverity,
```

```
                    CancelTypeStr[ monconfig.RMThresholdEventSeverity]);
      } else if ( monconfig.monitorType == rmLeakyBucket ) {

           printf("               Bucket: %u ", monconfig.RMbucketSize.rmValueU32
);
           printf("           Fill Value: %u ", monconfig.RMfillValue.rmValueU32 );
           printf( "        EventSeverity: %d (%s)\n",
                           monconfig.RMleakyBucketEventSeverity,
                           SeverityTypeStr[ monconfig.RMleakyBucketEventSeverity ]);
      } else {
              printf( "UNKNOWN Monitor Configuration type.\n");
      }

      return 0;
}

int getStatSize( rmStatisticKey statKey)
{
   rmStatisticInfo StatInfo;
   int result;
   if( (result= ResourceMonitor::rmGetStatisticInfo( statKey.SubsystemId,
                                    statKey.StatisticId,
                                    &StatInfo)) ) {
       perror( "GetStatisticInfo:");
       printf("\nresult=%d\n", result);
       printf( " SubsystemId=");
       RMuid uid(statKey.SubsystemId);
       uid.print(cout);
       printf( "   StatisticId=%u", statKey.StatisticId);
       cout << "\n";
       return -1;
   } else {
       return StatInfo.size;
   }
}

void   cleanup( int signum)
{

    ResourceMonitor::RMmonitor *newp;
    size_t Mcount = ResourceMonitor::rmGetMonitorCount();

    if( Mcount ) {
       rmMonitorInfo *MonBuffer = (rmMonitorInfo *)malloc(sizeof( rmMonitorInfo ) *
Mcount);
       rmMonitorInfo *pMon = MonBuffer;
       printf( "Removing SAMPLE monitors...\n");
       if( ResourceMonitor::rmGetMonitors( pMon, &Mcount ) != 0 ) {
             perror("cleanup(): rmGetMonitors()");
             free(MonBuffer);
             return;
       }
       for( size_t p=0; p < Mcount ; p++ ) {
           try {
                newp= new ResourceMonitor::RMmonitor( pMon[p].uid);
                string SampleName( newp->GetMonitorDescription( rmShortDescription
));
```

```
                if( SampleName.find("SAMPLE")== 0 ) {
                    char cUID[ MAX_GUID_STRLEN ];
                    RMuid tUID( pMon[p].uid );
                    tUID.unparse( cUID);
                    string SampleUID( cUID );
                    cout << "Clean up: " << SampleName << "\n";
                    cout << "          Delete query: ";
                    string ShellCmd("evlnotify --delete `evlnotify -l | grep "
                                    + SampleUID +
                                    " | awk -F: '{ print $1 }'`");
                    cout << ShellCmd << "\n";
                    system( ShellCmd.c_str() );
                    cout << "          Delete monitor: ";
//   cout << "Deleting monitor: (" << SampleUID << ") " << SampleName << "\n";
                    ShellCmd = "rmxml delete -i=" + SampleUID;
                    cout << ShellCmd << "\n";
                    system( ShellCmd.c_str() );
                }
            }
            catch(...) {
                    perror("Monitor not found: ");
                    RMuid toPrint( pMon[p].uid );
                    toPrint.print(cout);
                    cout << "\n";
            }

       }
       free( MonBuffer);
    }
    fflush( stdout);
    exit(0);

}
```

Appendix – Sample Subsystem (C++ required)

```
/***************************************************************************
                          RM_SampleSubsystem.h  -  description
                             -------------------
    begin                : Mon May 13 13:25:15 PDT 2002
    copyright            : (C) 2002 by Zhu Yi
    email                : yi.zhu@intel.com
 ***************************************************************************/


//  Copyright Intel Corp. 2002 yi.zhu@Intel.com

/***************************************************************************
 *                                                                         *
 * Recipient has requested a license and Intel Corporation (Intel) is      *
 * willing to grant a license for the software entitled Resource Monitor   *
 * Facility (the Software) being provided by Intel Corporation.            *
 *                                                                         *
 * The following definitions apply to this License:                        *
 *                                                                         *
 * Licensed Patents means patent claims licensable by Intel Corporation    *
 * which are necessarily infringed by the use or sale of the Software      *
 * alone or when combined with the operating system referred to below.     *
 * Recipient means the party to whom Intel delivers this Software.         *
 * Licensee means Recipient and those third parties that receive a license *
 * to any operating system available under the GNU Public License version  *
 * 2.0 or later.                                                           *
 *                                                                         *
 * Copyright (c) 1996-2001 Intel Corporation                               *
 * All rights reserved.                                                    *
 *                                                                         *
 * The license is provided to Recipient and Recipients Licensees under the *
 * following terms.                                                        *
 *                                                                         *
 * Redistribution and use in source and binary forms of the Software, with *
 * or without modification, are permitted provided that the following      *
 * conditions are met:                                                     *
 *                                                                         *
 * Redistributions of source code of the Software may retain the above     *
 * copyright notice, this list of conditions and the following disclaimer. *
 * Redistributions in binary form of the Software may reproduce the above  *
 * copyright notice, this list of conditions and the following disclaimer  *
 * in the documentation and/or other materials provided with the           *
 * distribution.                                                           *
 * Neither the name of Intel Corporation nor the names of its contributors *
 * shall be used to endorse or promote products derived from this Software *
 * without specific prior written permission.                              *
 *                                                                         *
 * Intel hereby grants Recipient and Licensees a non-exclusive, worldwide, *
 * royalty-free patent license under Licensed Patents to make, use, sell,  *
 * offer to sell, import and otherwise transfer the Software, if any, in   *
 * source code and object code form. This license shall include changes to *
 * the Software that are error corrections or other minor changes to the   *
 * Software that do not add functionality  or features when the Software   *
```

```
 * is incorporated in any version of a operating system that has been      *
 * distributed under the GNU General Public License 2.0 or later.  This    *
 * patent license shall apply to the combination of the Software and any   *
 * operating system licensed under the GNU Public License version 2.0 or   *
 * later if, at the time Intel provides the Software to Recipient, such    *
 * addition of the Software to the then publicly available versions of     *
 * such operating system available under the GNU Public License version    *
 * 2.0 or later (whether in gold, beta or alpha form) causes such          *
 * combination to be covered by the Licensed Patents. The patent license   *
 * shall not apply to any other combinations which include the Software.   *
 * No hardware per se is licensed hereunder.                               *
 *                                                                         *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS AS  *
 * IS AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,*
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR  *
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL INTEL OR ITS CONTRIBUTORS BE  *
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR     *
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF    *
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR         *
 * BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,   *
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR *
 * OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF  *
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.                              *
 *                                                                         *
 ***************************************************************************/


#ifndef __RM_SampleSubsystem_h__
#define __RM_SampleSubsystem_h__

#ifdef __cplusplus
/**
 * \namespace SampleSubsystem
 * \brief  SampleSubsystem contains a sample subsystem which can be
 * used to check if the resource monitor is running correctly.
 *
 * SampleSubsystem is the name space for the sample resource monitor subsystem.
 * \sa RM_SAMPLE_UUID_STRING RM_SAMPLE_NAME  RM_SAMPLE_DESCRIPTION
RM_SAMPLE_RESOURCE_NAME
 * RM_SAMPLE_RESOURCE_DESCRIPTION /usr/include/ResourceMonitor/RM_SampleSubsystem.h
 */
namespace SampleSubsystem
{
#endif // __cplusplus

#define   RM_SAMPLE_UUID_STRING   "cc0d1a5b-e060-4470-a5ef-1f503dd8263d"

//
// The text strings can be localized here and the library rebuilt
//

#define RM_SAMPLE_NAME         "SampleSubsystem"
#define RM_SAMPLE_DESCRIPTION       "SampleSubsystem is a very simple resource
monitor subsystem used only as a template."

#define RM_SAMPLE_RESOURCE_NAME     "SampleSubsystem Resource"
#define RM_SAMPLE_RESOURCE_DESCRIPTION "The SampleSubsystem has only one resource."
```

```
#define TJ_GAUGE_NAME     "Static Gauge"
#define TJ_GAUGE_DESCRIPTION   "A static gauge statistic"
#define TJ_GAUGE_UNIT     "Gauge unit"
#define TJ_GAUGE_TYPE     rmGauge
#define TJ_GAUGE_SIZE               rmSizeS32
#define TJ_GAUGE_SCALE              rmScaleNone
#define TJ_GAUGE_ID       9000

#define TJ_COUNTER_NAME      "Static Counter"
#define TJ_COUNTER_DESCRIPTION "A static counter statistic"
#define TJ_COUNTER_UNIT      "Counter unit"
#define TJ_COUNTER_TYPE      rmCounter
#define TJ_COUNTER_SIZE             rmSizeS32
#define TJ_COUNTER_SCALE            rmScaleNone
#define TJ_COUNTER_ID     9001

#define TJ_GAUGE_INC_NAME    "Increasing Gauge"
#define TJ_GAUGE_INC_DESCRIPTION "A gauge statistic that increases by 1 on every
read"
#define TJ_GAUGE_INC_UNIT    "Gauge unit"
#define TJ_GAUGE_INC_TYPE    rmGauge
#define TJ_GAUGE_INC_SIZE           rmSizeS32
#define TJ_GAUGE_INC_SCALE          rmScaleNone
#define TJ_GAUGE_INC_ID      9002

#define TJ_COUNTER_INC_NAME    "Increasing Counter"
#define TJ_COUNTER_INC_DESCRIPTION  "A counter statistic that increases by 1 on
every read"
#define TJ_COUNTER_INC_UNIT    "Counter unit"
#define TJ_COUNTER_INC_TYPE    rmCounter
#define TJ_COUNTER_INC_SIZE         rmSizeS32
#define TJ_COUNTER_INC_SCALE        rmScaleNone
#define TJ_COUNTER_INC_ID   9003

#ifdef InlineSampleSubsystem
#define RM_SAMPLE__MONITOR_NAME     "Inline Sample Monitor"
#define RM_SAMPLE__MONITOR_DESCRIPTION  "Inline Sample Monitor - for testing inline
monitor purposes only"
#endif //InlineSampleSubsystem

#ifdef __cplusplus
}; //SampleSubsystem
#endif
#endif //  __RM_SampleSubsystem_h__




/***************************************************************************
/***************************************************************************

/***************************************************************************
                          RM_SampleSubsystem.cpp  -  description
                             -------------------
    begin                : Tue May 14 16:31:41 PDT 2002
    copyright            : (C) 2001 by Zhu Yi
```

```
    email                : yi.zhu@intel.com
 ***************************************************************************/


//  Copyright Intel Corp. 2001 yi.zhu@Intel.com

/***************************************************************************
 *                                                                         *
 * Recipient has requested a license and Intel Corporation (Intel) is      *
 * willing to grant a license for the software entitled Resource Monitor   *
 * Facility (the Software) being provided by Intel Corporation.            *
 *                                                                         *
 * The following definitions apply to this License:                        *
 *                                                                         *
 * Licensed Patents means patent claims licensable by Intel Corporation    *
 * which are necessarily infringed by the use or sale of the Software      *
 * alone or when combined with the operating system referred to below.     *
 * Recipient means the party to whom Intel delivers this Software.         *
 * Licensee means Recipient and those third parties that receive a license *
 * to any operating system available under the GNU Public License version  *
 * 2.0 or later.                                                           *
 *                                                                         *
 * Copyright (c) 1996-2001 Intel Corporation                               *
 * All rights reserved.                                                    *
 *                                                                         *
 * The license is provided to Recipient and Recipients Licensees under the *
 * following terms.                                                        *
 *                                                                         *
 * Redistribution and use in source and binary forms of the Software, with *
 * or without modification, are permitted provided that the following      *
 * conditions are met:                                                     *
 *                                                                         *
 * Redistributions of source code of the Software may retain the above     *
 * copyright notice, this list of conditions and the following disclaimer. *
 * Redistributions in binary form of the Software may reproduce the above  *
 * copyright notice, this list of conditions and the following disclaimer  *
 * in the documentation and/or other materials provided with the           *
 * distribution.                                                           *
 * Neither the name of Intel Corporation nor the names of its contributors *
 * shall be used to endorse or promote products derived from this Software *
 * without specific prior written permission.                              *
 *                                                                         *
 * Intel hereby grants Recipient and Licensees a non-exclusive, worldwide, *
 * royalty-free patent license under Licensed Patents to make, use, sell,  *
 * offer to sell, import and otherwise transfer the Software, if any, in   *
 * source code and object code form. This license shall include changes to *
 * the Software that are error corrections or other minor changes to the   *
 * Software that do not add functionality  or features when the Software   *
 * is incorporated in any version of a operating system that has been      *
 * distributed under the GNU General Public License 2.0 or later.  This    *
 * patent license shall apply to the combination of the Software and any   *
 * operating system licensed under the GNU Public License version 2.0 or   *
 * later if, at the time Intel provides the Software to Recipient, such    *
 * addition of the Software to the then publicly available versions of     *
 * such operating system available under the GNU Public License version    *
 * 2.0 or later (whether in gold, beta or alpha form) causes such          *
 * combination to be covered by the Licensed Patents. The patent license   *
```

```
 * shall not apply to any other combinations which include the Software.   *
 * No hardware per se is licensed hereunder.                               *
 *                                                                         *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS AS  *
 * IS AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,*
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR  *
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL INTEL OR ITS CONTRIBUTORS BE  *
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR     *
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF    *
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR         *
 * BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,   *
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR *
 * OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF  *
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.                              *
 *                                                                         *
 ***************************************************************************/
#define InSubsystemMonitor
#include <IRMRegistration.h>
#include <Statistic.h>
#include <Monitor.h>
#include "RM_SampleSubsystem.h"
#include <stdio.h>
#include <syslog.h>
#include <string>
#include <unistd.h>
#include <sys/time.h>
#include <sys/sysinfo.h>
#include <dirent.h>
#include "stl.h"
#include <map>

namespace SampleSubsystem
{

enum RM_SampleStats
{
  STAT_none,
  STAT_gauge,
  STAT_counter,
  STAT_gauge_inc,
  STAT_counter_inc
};

typedef struct {
  rmString            name;
  rmString            description;
  rmString            units;
  rmStatisticType     type;
      rmStatisticSize     size;
  rmStatisticScale    scale;
  rmID                id;
  enum RM_SampleStats stat;
} TSRegister;

#define GaugeENTRY { TJ_GAUGE_NAME, TJ_GAUGE_DESCRIPTION, TJ_GAUGE_UNIT,
TJ_GAUGE_TYPE, TJ_GAUGE_SIZE, TJ_GAUGE_SCALE, TJ_GAUGE_ID, STAT_gauge }
```

```
#define CounterENTRY  { TJ_COUNTER_NAME, TJ_COUNTER_DESCRIPTION, TJ_COUNTER_UNIT,
TJ_COUNTER_TYPE, TJ_COUNTER_SIZE, TJ_COUNTER_SCALE, TJ_COUNTER_ID, STAT_counter }
#define GaugeIncENTRY { TJ_GAUGE_INC_NAME, TJ_GAUGE_INC_DESCRIPTION,
TJ_GAUGE_INC_UNIT, TJ_GAUGE_INC_TYPE, TJ_GAUGE_INC_SIZE, TJ_GAUGE_INC_SCALE,
TJ_GAUGE_INC_ID, STAT_gauge_inc }
#define CounterIncENTRY { TJ_COUNTER_INC_NAME, TJ_COUNTER_INC_DESCRIPTION,
TJ_COUNTER_INC_UNIT, TJ_COUNTER_INC_TYPE, TJ_COUNTER_INC_SIZE, TJ_COUNTER_INC_SCALE,
TJ_COUNTER_INC_ID, STAT_counter_inc }

const size_t RM_SAMPLE_STATISTICS = 4;
// maintain case statement in StatisticFactory also when adding or deleting stats
const struct {
   RMuid      guid;
   TSRegister subsystem;
   TSRegister resource;
   TSRegister statistics[RM_SAMPLE_STATISTICS];
#ifdef InlineSampleSubsystem
   TSRegister monitor;
#endif
} g_registrationData = {
  ( RM_SAMPLE_UUID_STRING ),
  { RM_SAMPLE_NAME, RM_SAMPLE_DESCRIPTION, 0, (rmStatisticType)0,
(rmStatisticSize)0, (rmStatisticScale)0, 0, STAT_none },
  { RM_SAMPLE_RESOURCE_NAME, RM_SAMPLE_RESOURCE_DESCRIPTION, 0, (rmStatisticType)0,
(rmStatisticSize)0, (rmStatisticScale)0, 0, STAT_none },
  { GaugeENTRY, CounterENTRY, GaugeIncENTRY, CounterIncENTRY }
#ifdef InlineSampleSubsystem
 ,{ RM_SAMPLE_MONITOR_NAME, RM_SAMPLE_MONITOR_DESCRIPTION, 0, (rmStatisticType)0,
(rmStatisticSize)0, (rmStatisticScale)0, 0, STAT_none }
#endif
};


class CSubsystemInfo : public SubsystemMonitor::ISubsystemMonitor
{
public:
   CSubsystemInfo::CSubsystemInfo(RMuid guid);
   CSubsystemInfo::~CSubsystemInfo();

   int getSubsystemInterfaceVersion();
   rmSubsystemInfo    *getSubsystemInfo();
   rmResourceInfo   *getResourceInfo(size_t index);
   rmStatisticInfo    *getStatisticInfo(size_t index);
   //rmInlineMonitorInfo *getInlineMonitorInfo(rmID statid, size_t index);

private:
   RMuid m_guid;
   rmSubsystemInfo    m_subInfo;
   rmResourceInfo   m_resourceInfo;
   rmStatisticInfo    m_statisticInfo[RM_SAMPLE_STATISTICS];
#ifdef InlineSampleSubsystem
   //rmInlineMonitorInfo m_monitorInfo[RM_SAMPLE_STATISTICS];
#endif // InlineSampleSubsystem
};

CSubsystemInfo::CSubsystemInfo(RMuid a_guid)
{
```

```
  if( a_guid == g_registrationData.guid ) {
    m_subInfo.name = g_registrationData.subsystem.name;
    m_subInfo.description = g_registrationData.subsystem.description;
    m_subInfo.id = a_guid.id;
    m_subInfo.noResources = 1;
    m_subInfo.noStatistics = RM_SAMPLE_STATISTICS;
    m_resourceInfo.name = g_registrationData.resource.name;
    m_resourceInfo.description = g_registrationData.resource.description;
    m_resourceInfo.SubsystemId = a_guid.id;
    m_resourceInfo.id = g_registrationData.resource.id;
    for( size_t i = 0 ; i < RM_SAMPLE_STATISTICS ; i++ ) {
      m_statisticInfo[i].SubsystemId = a_guid.id;
      m_statisticInfo[i].id = g_registrationData.statistics[i].id;
      m_statisticInfo[i].name = g_registrationData.statistics[i].name;
      m_statisticInfo[i].description =
g_registrationData.statistics[i].description;
      m_statisticInfo[i].units = g_registrationData.statistics[i].units;
      m_statisticInfo[i].type = g_registrationData.statistics[i].type;
      m_statisticInfo[i].size = g_registrationData.statistics[i].size;
      m_statisticInfo[i].scale= g_registrationData.statistics[i].scale;
#ifdef InlineSampleSubsystem
      m_statisticInfo[i].noInlineMonitors = 1;

      m_monitorInfo[i].name = g_registrationData.monitor.name;
      m_monitorInfo[i].description = g_registrationData.monitor.description;
      m_monitorInfo[i].config.monitorType = rmWatermarking;
      m_monitorInfo[i].config.typeConfiguration.watermark.type = rmDualWatermark;
      m_monitorInfo[i].config.statId.SubsystemId = a_guid.id;
      m_monitorInfo[i].config.statId.ResourceId = 0;
      m_monitorInfo[i].config.statId.StatisticId =
g_registrationData.statistics[i].id;
#else
      //m_statisticInfo[i].noInlineMonitors = 0;
#endif // InlineSampleSubsystem
    }
  }
}
CSubsystemInfo::~CSubsystemInfo(){}


int CSubsystemInfo::getSubsystemInterfaceVersion()
{
  return   RESOURCE_MONITOR_VERSION;
};

rmSubsystemInfo *CSubsystemInfo::getSubsystemInfo()
{
  return   &m_subInfo;
};

rmResourceInfo *CSubsystemInfo::getResourceInfo(size_t a_index)
{
  if( a_index == 0 ) return &m_resourceInfo;
  return 0;
};

rmStatisticInfo *CSubsystemInfo::getStatisticInfo(size_t a_index)
```

```
{
  if( a_index >= 0 && a_index < RM_SAMPLE_STATISTICS  ) return
&m_statisticInfo[a_index];
  return 0;
};

/*
rmInlineMonitorInfo *CSubsystemInfo::getInlineMonitorInfo(rmID a_id, size_t a_index)
{
#ifdef InlineSampleSubsystem
  if( a_index >= 0 && a_index < RM_SAMPLE_STATISTICS && a_id >= 0 && a_id <
RM_SAMPLE_STATISTICS )
    return &m_monitorInfo[a_index];
#endif // InlineSampleSubsystem
  return 0;
};
*/


static rmValue theValue;
class CSampleStatistic : public SubsystemMonitor::Statistic
{
public:
  CSampleStatistic(rmID a_statid);
  ~CSampleStatistic();

  int readValue(rmValue * val, time_t a_timestamp);
  int resetCounterValue(rmValue *val);
protected:
  rmID m_statid;
};

CSampleStatistic::CSampleStatistic(rmID a_statid)
{
  m_statid = a_statid;
}

CSampleStatistic::~CSampleStatistic()
{
}

int  CSampleStatistic::resetCounterValue(rmValue *a_val)
{
  theValue = *a_val;
  return 0;
  //return ENOSYS;
}

int  CSampleStatistic::readValue(rmValue * val, time_t a_timestamp)
{
  struct timevaltv;
  struct timezone  tz;

  if( a_timestamp == 0 ) {
    gettimeofday(&tv,&tz);
    a_timestamp = tv.tv_sec;
  }
```

```
   *val = theValue;
   if ( (m_statid == 9001) || (m_statid == 9003) ) {
     theValue.rmValueU32++;
   } else
     theValue.rmValueU32 = 9;

   return 0;
}


#ifdef InlineSampleSubsystem
//
//     In Line Monitor
//

class CSampleSubsystem  : public SubsystemMonitor::Monitor
{
public:
   CSampleSubsystem(RMuid a_guid, rmID a_resourceid, rmID a_statid, rmID a_monid);
   ~CSampleSubsystem();

   int startMonitor();
   int stopMonitor();
   int resetMonitor();
   int pauseNotification();
   int resetNotification();
   int getMonitorConfiguration(rmMonitorConfiguration *a_config);
   int setMonitorConfiguration(rmMonitorConfiguration &a_config);
   int checkMonitor(rmMonitorState *a_state);
protected:
private:
   rmID m_monid;
   rmMonitorConfiguration m_config;

};

CSampleSubsystem::CSampleSubsystem(RMuid a_guid, rmID a_resourceid, rmID a_statid,
rmID a_monid)
{
   m_config.monitorType = rmWatermarking;
   m_config.typeConfiguration.watermark.type = rmDualWatermark;
   m_config.statId.SubsystemId = a_guid.id;
   m_config.statId.ResourceId = a_resourceid;
   m_config.statId.StatisticId = a_statid;
   m_monid = a_monid;
};

CSampleSubsystem::~CSampleSubsystem(){}

int  CSampleSubsystem::startMonitor()
{
   return 0; // TODO //Sample inline monitor
};

int  CSampleSubsystem::stopMonitor()
{
   return 0; // TODO //Sample inline monitor
```

```
};

int  CSampleSubsystem::resetMonitor()
{
  return 0; // TODO //Sample inline monitor
};

int  CSampleSubsystem::pauseNotification()
{
  return 0; // TODO //Sample inline monitor
};

int  CSampleSubsystem::resetNotification()
{
  return 0; // TODO //Sample inline monitor
};

int CSampleSubsystem::getMonitorConfiguration(rmMonitorConfiguration *config)
{
  config = &m_config;
  return 0; // TODO //Sample inline monitor
};

int CSampleSubsystem::setMonitorConfiguration(rmMonitorConfiguration &config)
{
  return 0; // TODO //Sample inline monitor
};

int CSampleSubsystem::checkMonitor(rmMonitorState *state)
{
  return 0; // TODO //Sample inline monitor
};
#endif // InlineSampleSubsystem

SubsystemMonitor::ISubsystemMonitor * ISubsystemFactory(RMuid a_guid) {
  if( a_guid == g_registrationData.guid ) {
    SubsystemMonitor::ISubsystemMonitor * mp;
    try {
      mp = new CSubsystemInfo(a_guid);
    } catch ( bad_alloc ) {
      return 0 ;
    }
    return mp;
  }
  return 0;
}

SubsystemMonitor::Statistic * StatisticFactory(RMuid a_guid,
                                               rmID a_resourceid,
                                               rmID a_statid) {
  if( a_guid == g_registrationData.guid ) {
    for( size_t i = 0 ; i < RM_SAMPLE_STATISTICS ; i++ ) {
      if( g_registrationData.statistics[i].id == a_statid )
        return new CSampleStatistic (a_statid);
    }
  }
  return 0;
```

```
}
#ifdef InlineSampleSubsystem
SubsystemMonitor::Monitor *MonitorFactory(rmID a_mon,
                                          rmMonitorConfiguration a_config,
                                          RMuid a_monUID) {
  if( g_registrationData.guid == a_config.statId.SubsystemId && a_mon == 0) {
    return  new CSampleSubsystem(a_config.statId.SubsystemId,
ca_onfig.statId.ResourceId, a_config.statId.StatisticId, a_mon);
  }
  return 0;
}
#endif // InlineSampleSubsystem

#include <IRMRegistration.h>
class CSampleSubsystem
{
public:
  // this global constructor is required for successful registration with the
resource monitor daemon
  CSampleSubsystem()
  {

  SubsystemMonitor::IRMRegistration::registerSubsystem(SampleSubsystem::g_registrat
ionData.guid, &SampleSubsystem::ISubsystemFactory);

  SubsystemMonitor::IRMRegistration::registerStatistic(SampleSubsystem::g_registrat
ionData.guid, &SampleSubsystem::StatisticFactory);
#ifdef InlineSampleSubsystem

  SubsystemMonitor::IRMRegistration::registerMonitor(SampleSubsystem::g_registratio
nData.guid, &SampleSubsystem::MonitorFactory);
#endif // InlineSampleSubsystem
  }
};
// our one instance of the proxy
CSampleSubsystem g_registerSampleSubsystem;

} // SampleSubsystem
```

Appendix – References

Provided is a list of web links, documents, book references, etc. that are either referenced in this document or can provide more insight for the reader.

Resource Monitor Specific Sites:

http://carrierlinux.org

http://builds.carrierlinux.org/index.php

\\localhost\opt\resourcemon\doc\html\en\index.html

POSIX Event Log Specific Sites:

https://sourceforge.net/project/showfiles.php?group_id=34226

http://evlog.sourceforge.net

http://evlog.sourceforge.net/linuxEvlog.html

XML Specific Sites:

http://www.w3.org/XML

Appendix – Definitions

Define any terms or technologies discussed in the white paper that are not well known. Provide a link to more information on the topic, as well as a one-line summary of what can be found via that link.

A

Alarm - Physical indication, usually persistent, that an event has occurred.

B

Bidirectional Threshold - Gauge statistic monitor that triggers an event when a threshold is met or crossed in one direction with a "canceling" event being triggered when the threshold is crossed in the other direction.

C

Consumer Application - An application that uses the application interface provided by the Resource Monitoring Facility.

Counter - Statistic that can only increase its value.

D

Data Capture Subsystem - Subsystems for persisting statistic readings of a monitor.

Daemon Monitor - A monitor implemented by the Resource Monitor's daemon process that tests a statistic's condition within the RM daemon process, usually by periodically polling a subsystem's statistic.

E

Event – A phenomenon of interest that occurs in a system. An Event Indication, or Event Record, is a record of an event, such as a log entry or inter-process message. Often the terms, “event”, “event record”, “alert”, and “alarm” are used interchangeably, even though there are nuances in the definitions of each.

Event Log - System and application event logging facility with a notification feature.

F

G

Gauge - Statistic that can freely move up or down.

Inline Monitor - A monitor implemented by a subsystem that tests a statistic's condition where the statistic is being maintained.

H

I

J

K

L

Leaky Bucket Monitor - Monitors the rate of increase of a counter statistic. The rate of counter increase is measured in a sliding window of time where a window period = bucket size / fill value * sampling interval.

M

N

O

P

Q

R

Resource - An entity defined by the subsystem from which statistics can be queried.

S

Stateful Threshold - A threshold monitor logs an event record when a threshold is met or crossed in one direction. The stateful monitor continues to log an event record as long as the condition persists. A threshold monitor that is not stateful only logs the event once.

Statistics - Queriable numeric data.

Subsystem - A dynamically loadable library that provides access methods used by the daemon to query statistic values. When statistics and inline monitors are in kernel modules, the library acts as a proxy for the subsystem. A subsystem is considered instrumented when a loadable library is installed providing the access methods for the daemon.

T

Threshold Alert - An event indication triggered by a threshold condition. The alert is an event record logging in the event log. Threshold Alerts can have the following POSIX-defined severities:

U

V

W

Watermark - Maximum or minimum gauge value for a given period.

X

Y

Z

Appendix – Abbreviations and Acronyms

Abbreviation	Description