ApMon User Guide

December 23, 2005

Chapter 1

ApMon General User Guide

1.1 ApMon - General User Guide

1.1.1 ApMon Initialization

There are several ways to initialize ApMon:

A first method to initialize ApMon is from a configuration file, which contains the IP addresses or DNS names of the hosts running MonALISA, to which the data will be sent; the ports on which the MonALISA services listen on the destination hosts should also be specified in the file. The configuration file also contains lines that specify lines for configuring xApMon (see Section 1.1.3). The lines that specify the destination hosts have the following syntax:

```
IP_address|DNS_name[:port] [password]
```

Examples:

```
rb.rogrid.pub.ro:8884 mypassword
rb.rogrid.pub.ro:8884
ui.rogrid.pub.ro mypassword
ui.rogrid.pub.ro
```

If the port is not specified, the default value 8884 will be assumed. If the password is not specified, an empty string will be sent as password to the MonALISA host (and the host will accept the datagram either if it does not require a password for the ApMon packets or if the machine from which the packet was sent is in the host's "accept" list). The configuration file may contain blank lines and comment lines (starting with "#"); these lines are ignored, and so are the leading and the trailing white spaces from the other lines.

Another method to initialize ApMon is to provide a list which contains hostnames and ports as explained above, and/or URLs; the URLs point to plain text configuration files which have the format described above. The URLs may also represent requests to a servlet or a CGI script which can automatically provide the best configuration, taking into account the geographical zone in which the machine which runs ApMon is situated, and the application for which ApMon is used. The geographical zone is determined from the machine's IP and the application name is given by the user as the value of the "appName" parameter included in the URL.

1.1.2 Sending Datagrams with User Parameters

A datagram sent to the MonaLisa module has the following structure:

- a header which has the following syntax:

  ```
  v:<ApMon_version>p:<password>
  ```

  (the password is sent in plaintext; if the MonALISA host does not require a password, a 0-length string should be sent instead of the password).
- cluster name (string) - the name of the monitored cluster
- node name (string) - the name of the monitored nodes
- number of parameters (int)
- for each parameter: name (string), value type (int), value (can be double, int or string)
- optionally a timestamp (int) if the user wants to specify the time associated with the data; if the timestamp is not given, the current time on the destination host which runs MonALISA will be used. The option to include a timestamp is possible since version 2.0.

The configuration file and/or URLs can be periodically checked for changes, in a background thread or process, but this option is disabled by default. It can be enabled from the configuration file as follows:

- to enable/disable the periodical checking of the configuration file or URLs:

  ```
  xApMon_conf_recheck = on/off
  ```

- to set the time interval at which the file/URLs are checked for changes:

  ```
  xApMon_recheck_interval = number_of_seconds
  ```

1.1.3 xApMon - Automatically Sending Monitoring Information

ApMon can be configured to send automatically, in a background thread, monitoring information regarding the application or the system. The system information is obtained from the *proc/* filesystem and the job information is obtained by parsing the output of the ps command. If job monitoring for a process is requested, all its sub-processes will be taken into consideration (i.e., the resources consumed by the process and all the subprocesses will be summed).
There are three categories of monitoring datagrams that ApMon can send:
a) job monitoring information - contains the following parameters:

- *run_time*: elapsed time from the start of this job
- *cpu_time*: processor time spent running this job
- *cpu_usage*: percent of the processor used for this job, as reported by *ps*
- *virtualmem*: virtual memory occupied by the job (in KB)
- *rss*: resident image size of the job (in KB)
- *mem_usage*: percent of the memory occupied by the job, as reported by *ps*
- *workdir_size*: size in MB of the working directory of the job

- *disk_total*: size in MB of the disk partition containing the working directory
- *disk_used*: size in MB of the used disk space on the partition containing the working directory
- *disk_free*: size in MB of the free disk space on the partition containing the working directory
- *disk_usage*: percent of the used disk partition containing the working directory

b) system monitoring information - contains the following parameters:

- *cpu_usr*: percent of the time spent by the CPU in user mode
- *cpu_sys*: percent of the time spent by the CPU in system mode
- *cpu_nice*: percent of the time spent by the CPU in nice mode
- *cpu_idle*: percent of the time spent by the CPU in idle mode
- *cpu_usage*: CPU usage percent
- *pages_in*: the number of pages paged in per second (average for the last time interval)
- *pages_out*: the number of pages paged out per second (average for the last time interval)
- *swap_in*: the number of swap pages brought in per second (average for the last time interval)
- *swap_out*: the number of swap pages brought out per second (average for the last time interval)
- *load1*: average system load over the last minute
- *load5*: average system load over the last 5 min
- *load15*: average system load over the last 15 min
- *mem_used*: amount of currently used memory, in MB
- *mem_free*: amount of free memory, in MB
- *mem_usage*: used system memory in percent
- *swap_used*: amount of currently used swap, in MB
- *swap_free*: amount of free swap, in MB
- *swap_usage*: swap usage in percent
- *net_in*: network (input) transfer in kBps
- *net_out*: network (input) transfer in kBps
- *net_errs*: number of network errors (these will produce params called sys_ethX_in, sys_ethX_out, sys_ethX_errs, corresponding to each network interface)
- *processes*: curent number of processes (this will also produce parameters called processes_{D,R,T,S,Z}- number of processes in the D (uninterruptible sleep),R (running), T(traced/stopped), S (sleeping), Z (zombie) states)
- *uptime*: system uptime in days
- *net_sockets*: the number of open TCP, UDP, ICM, Unix sockets (this will produce parameters called sockets_tcp, sockets_udp, ...)
- *net_tcp_details*: the number of TCP sockets in each possible state (this will produce parameters called sockets_tcp_ESTABLISHED, sockets_TCP_LISTEN, ...)

c) general system information - contains the following parameters:

- *hostname*: the machine's hostname
- *ip*: will produce ethX_ip params for each interface

- *cpu_MHz*: CPU frequency
- *no_CPUs*: number of CPUs
- *total_mem*: total amount of memory, in MB
- *total_swap*: total amount of swap, in MB

The parameters can be enabled/disabled from the configuration file (if they are disabled, they will not be included in the datagrams). In order to enable/disable a parameter, the user should write in the configuration file lines of the following form:

```
xApMon_job_parametername = on/off
```

(for job parameters) or:

```
xApMon_sys_parametername = on/off
```

(for job parameters) or:

```
xApMon_parametername = on/off
```

(for general system parameters) Example:

```
xApMon_job_run_time = on
xApMon_sys_load1 = off
xApMon_no_CPUs = on
```

By default, all the parameters are enabled.

The job/system monitoring can be enabled/disabled by including the following lines in the configuration file:

```
xApMon_job_monitoring = on/off
xApMon_sys_monitoring = on/off
```

The datagrams with general system information are only sent if system monitoring is enabled, at greater time intervals than the system monitoring datagrams. To enable/disable the sending of general system information datagrams, the following line should be written in the configuration file:

```
xApMon_general_info = on/off
```

The time interval at which job/system monitoring datagrams are sent can be set with:

```
xApMon_job_interval = number_of_seconds
xApMon_sys_interval = number_of_seconds
```

Chapter 2

ApMon C User Guide

2.1 Installation

To compile the ApMon routines and all the examples, type:

```
./configure [options]
make
make install
```

where ”options” are the typical configure options. The library will be installed in *$prefix/lib* and the *ApMon.h* include file into the *$prefix/include* directory.

If you have Doxygen, you can get the API docs by issuing make doc.

2.2 Using ApMon

2.2.1 ApMon Initialization

In the C version, the ApMon features are available as functions that operate on a structure called ApMon. An ApMon structure can be initialized with one of the functions (see the API docs for more details):

```
ApMon* apMon_init(char *filename);
```

- initializes ApMon from a configuration file or webpage, whose name/URL is given as parameter

```
ApMon *apMon_stringInit(char *destinationsList);
```

- initializes ApMon from a list which contains destination hosts, specified as ”address:[port][password]”, and/or configuration webpages; the items of list are placed in a string, separated by commas

```
ApMon *apMon_arrayInit(int nDestinations, char **destAddresses,
                       int *destPorts,    char **destPasswds );
```

- initializes ApMon from a list of destination hosts and the correspondong lists of ports and passwords

2.2.2 Sending Datagrams

There are two ways in which the user can send parameter values to MonALISA:

a a single parameter in a datagram

b multiple parameters in a datagram

For sending a datagram with a single parameter, the user should call the function `apMon_sendParameter()` which has several overloaded variants.

For sending multiple parameters in a datagram, the user should call the function `apMon_sendParameters()`, which receives as arguments arrays with the names and the values of the parameters to be sent.

Since version 2.0 there are two additional functions, `apMon_sendTimedParameter()` and `apMon_sendTimedParame` which allow the user to specify a timestamp for the parameters.

ApMon C does not send parameters whose names are NULL strings or string parameters that have NULL value (these parameters are ”skipped”).

2.2.3 Configuring ApMon with the aid of the API

The behaviour of ApMon can be configured not only from the configuration files or webpages, but also with the aid of the API.

In order to enable the periodical reloading of the configuration files, the user should call `apMon_setConfRecheck_d();` the value of the time interval at which the recheck operatins are performed, can be set with the functions `apMon_setConfRecheck()` or `apMon_setRecheckInterval()`.

To enable/disable the automated job/system monitoring, and also to set the time intervals, the functions `apMon_setJobMonitoring()` and `apMon_setSysMonitoring()` can be used.

2.2.4 Automated Job Monitoring

To monitor jobs, you have to specify the PID of the parent process for the tree of processes that you want to monitor, the working directory, the cluster and the node names that will be registered in MonALISA (and also the job monitoring must be enabled). If work directory is ””, no information will be retrieved about disk:

```
apMon_addJobToMonitor(ApMon *apm, long pid, char *workdir,
                      char *clusterName, char *nodeName);
```

To stop monitoring a job, the `removeJobToMonitor(long pid)` should be called.

2.2.5 Logging

ApMon prints its messages to the standard output, with the aid of the `logger()` function from *utils.c* (see the API documentation). The user may print its own messages with this function (see *example_1.cpp*, *example_2.cpp*). Each message has a level which represent its importance. The possible levels are FATAL, WARNING, INFO, FINE, DEBUG. Only the messages which have a level with greater or equal importance than the current ApMon loglevel are printed. The ApMon loglevel can be set from the configuration file (by default it is INFO):

```
xApMon_loglevel = <level>
```

e.g.,

```
xApMon_loglevel = FINE
```

The ApMon loglevel can also be set with the `apMon_setLogLevel()` function.

For a better understanding of how to use the functions mentioned above, see the Doxygen documentation and the examples.

2.2.6 Restrictions

The following values are limited to some constants defined in ApMon.h:

- the maximum number of destinations to which the datagrams can be sent (specified by the constant MAX_N_DESTINATIONS; by default it is 30)
- the maximum size of a datagram (specified by the constant MAX_DGRAM_SIZE; by default it is 8192B and the user should not modify this value as some hosts may not accept larger datagrams)
- the password may have at most 20 characters
- the maximum number of jobs that can be monitored is 30
- the maximum number of messages that can be sent per second, on average, is limited, in order to avoid the accidental growth of the network load (which may happen, for example, if the user places the sendParameters() calls in a loop, without pauses between them). To set the maxim number of messages that can be sent per second by ApMon, you can use the function `apMon_setMaxMsgRate(ApMon *apm, int rate)`. Another way to set the maximum number of messages is to specify it in the configuration file:

  ```
  xApMon_maxMsgRate = 30
  ```

  By default, the maximum number of messages per second is 50.

2.3 How to Write a Simple C Program with ApMon

In this sect1 we show how the ApMon API can be used to write a simple program that sends monitoring datagrams. The source code for this short tutorial (slightly modified) can be found in the *example_2.c* file under the *examples/* directory.

The program generates some double values for a parameter called "my_cpu_load" and sends them to the MonALISA destinations. The number of iterations is given as a command line argument; in each iteration two datagrams are sent, one with timestamp and one without timestamp.

With this example program we'll illustrate the steps that should usually be taken to write a program with ApMon:

1. Include the ApMon headers (and possibly other necessary headers):

```
#include <stdlib.h>
#include <time.h>

#include "ApMon.h"
```

2. Initialize the variables we shall use...

```
int main(int argc, char **argv) {
char *destinationsList = "rb.rogrid.pub.ro password,
                          http://lcfg.rogrid.pub.ro/~corina/destinations_2.conf";
int nDatagrams = 20;
ApMon *apm;
double val;
int i, ret, timestamp;
srand(time(NULL));

if (argc ==2)
    nDatagrams = atoi(argv[1]);
```

3. Initialize the ApMon structure (in this example we used an intialization list containing the name of a destination host, rb.rogrid.pub.ro, and a webpage where other destination hosts and possibly ApMon options can be specified).

```
/* initialize the ApMon structure */
apm = apMon_stringInit(destinationsList);
if (apm == NULL)
apMon_errExit("\nError initializing the ApMon structure");
```

4. Adjust ApMon's settings, if necessary (here we set the time interval for reloading the configuration page to 300 sec). This can also be done from the configuration file.

```
apMon_setRecheckInterval(apm, 300);
nDatagrams = atoi(argv[1]);
```

5. Send datagrams; we used here two kinds of functions: one that includes a timestamp in the datagram and one that doesn't.

```
for (i = 0; i < nDatagrams; i++) {
    /* add a value for the parameter (random between 0 and 2) */
    val = 2 * (double)rand() / RAND_MAX;
    printf("Sending %lf for cpu_load\n", val);
    /* use the wrapper function with simplified syntax */
    /* (the node name is left NULL, so the local IP will be sent instead) */
    ret = apMon_sendParameter(
                apm,
                "TestCluster2_c",
                NULL,
                "my_cpu_load",
                XDR_REAL64,
                (char *)&val
          );
```

6. For each datagram, check that it was send successfully. The ApMon functions return RET_SUCCESS (0) on success and RET_ERROR (-1) if an error occured.

```
    if (ret != RET_SUCCESS) {
        apMon_free(apm);
        fprintf(stderr, "\nError sending result");
    }

    /* now send the datagram with a timestamp */
    timestamp = time(NULL) - (5 * 3600); /* as if it was sent 5 hours ago */
    ret = apMon_sendTimedParameter(
                apm,
                "TestCluster2_c_5",
                NULL,
                "my_cpu_load",
                XDR_REAL64,
                (char *)&val,
                timestamp
          );
    if (ret != RET_SUCCESS) {
        apMon_free(apm);
        fprintf(stderr, "\nError sending result");
    }

sleep(2);
}
```

7. Destroy the ApMon structure:

```
apMon_free(apm);
return 0;
}
```

Chapter 3

ApMon C++ User Guide

3.1 Installation

To compile the ApMon routines and all the examples, type:

```
./configure [options]
make
make install
```

where ”options” are the typical configure options. The library will be installed in *$prefix/lib* and the *ApMon.h* include file into the *$prefix/include* directory.

If you have Doxygen, you can get the API docs by issuing `make doc`.

3.2 Using ApMon

3.2.1 ApMon Initialization

In the C++ version, the ApMon features are available as methods of a class called `ApMon`. An ApMon object can be initialized with one of the constructors (see the API docs for more details):

```
ApMon(char *initsource);
```

- initializes ApMon from a configuration file or webpage, whose name/URL is given as parameter

```
ApMon(int nDestinations, char **destinationsList);
```

- initializes ApMon from a list which contains strings of the form ”address:[port][password]” specifying destination hosts, and/or addresses of configuration webpages

```
ApMon(int nDestinations, char **destAddresses, int *destPorts, char **destPasswds)
```

- initializes ApMon from a list of destination hosts and the corresponding lists of ports and passwords

3.2.2 Sending Datagrams

There are two ways in which the user can send parameter values to MonALISA:

a a single parameter in a datagram

b multiple parameters in a datagram

For sending a datagram with a single parameter, the user should call the function `sendParameter()` which has several overloaded variants.

For sending multiple parameters in a datagram, the user should call the function `sendParameters()`, which receives as arguments arrays with the names and the values of the parameters to be sent.

Since version 2.0 there are two additional functions, `sendTimedParameter()` and `sendTimedParameters()`, which allow the user to specify a timestamp for the parameters.

ApMon C++ does not send parameters whose names are NULL strings or string parameters that have NULL value (these parameters are "skipped").

3.2.3 Configuring ApMon with the Aid of the API

The behaviour of ApMon can be configured not only from the configuration files or webpages, but also with the aid of the API.

In order to enable the periodical reloading of the configuration files, the user should call setConfCheck(true); the value of the time interval at which the recheck operatins are performed, can be set with the functions `setConfRecheck()` or `setRecheckInterval()`.

To enable/disable the automated job/system monitoring, and also to set the time intervals, the functions `setJobMonitoring()` and `setSysMonitoring()` can be used.

3.2.4 Automated Job Monitoring

To monitor jobs, you have to specify the PID of the parent process for the tree of processes that you want to monitor, the working directory, the cluster and the node names that will be registered in MonALISA (and also the job monitoring must be enabled). If work directory is "", no information will be retrieved about disk:

```
addJobToMonitor(long pid, char *workdir, char *clusterName, char *nodeName);
```

To stop monitoring a job, the `removeJobToMonitor(long pid)` should be called.

3.2.5 Logging

ApMon prints its messages to the standard output, with the aid of the `logger()` function from utils.cpp (see the API documentation). The user may print his own messages with this function (see *example_1.cpp*, *example_2.cpp* from the *examples/* directory). Each message has a level which represents its importance. The possible levels are FATAL, WARNING, INFO, FINE, DEBUG. Only the messages which have a level with greater or equal importance than the current ApMon loglevel are printed. The ApMon loglevel can be set from the configuration file (by default it is INFO):

```
xApMon_loglevel = <level>
```

e.g.,

```
xApMon_loglevel = FINE
```

The ApMon loglevel can also be set with the `setLogLevel()` function.

For a better understanding of how to use the functions mentioned above, see the Doxygen documentation and the examples.

3.2.6 Restrictions

The following values are limited to some constants defined in *ApMon.h*:

- the maximum size of a datagram (specified by the constant MAX_DGRAM_SIZE; by default it is 8192B and the user should not modify this valueas there are hosts that may not accept larger datagrams)
- the maximum number of destinations to which the datagrams can be sent (specified by the constant MAX_N_DESTINATIONS; by default it is 30)
- the password may have at most 20 characters
- the maximum number of jobs that can be monitored is 30
- the maximum number of messages that can be sent per second, on average, is limited, in order to avoid the accidental growth of the network load (which may happen, for example, if the user places the `sendParameters()` calls in a loop, without pauses between them). To set the maxim number of messages that can be sent per second by ApMon, you can use the function `setMaxMsgRate(int rate)`. Another way to set the maximum number of messages is to specify it in the configuration file:

  ```
  xApMon_maxMsgRate = 30
  ```

If you want to use features that involve the background thread (periodical configuration reloading, job/system monitoring), the ApMon object used must be alloc'ed dynamically (as seen in the examples).

3.3 How to Write a Simple C++ Program with ApMon

In this sect1 we show how the ApMon API can be used to write a simple program that sends monitoring datagrams. The source code for this short tutorial (slightly modified) can be found in the *example_2.cpp* file under the *examples/* directory.

The program generates some double values for a parameter called ”my_parameter” and sends them to the MonALISA destinations. The number of iterations is given as a command line argument; in each iteration two datagrams are sent, one with timestamp and one without timestamp.

With this example program we'll illustrate the steps that should usually be taken to write a program with ApMon:

1. Include the ApMon headers (and possibly other necessary headers):

```
#include <time.h>

#include "ApMon.h"

/* the next two lines are necessary for using the logging facility */
#include "utils.h"
using namespace apmon_utils;
```

2. Initialize the variables we shall use...

```
int main(int argc, char **argv) {
char *destList[2]= {"http://lcfg.rogrid.pub.ro/~corina/destinations_2.conf",
                    "rb.rogrid.pub.ro password"};
int nDatagrams = 20;
char logmsg[100];
double val = 0;
int i, timestamp;

if (argc ==2)
    nDatagrams = atoi(argv[1]);
```

3. Construct an ApMon object (in this example we used an intialization list containing the name of a destination host, *rb.rogrid.pub.ro,* and a webpage where other destination hosts and possibly ApMon options can be specified). The ApMon functions throw exceptions if errors appear, so it is recommended to place them in a try-catch block:

```
try {
    ApMon *apm = new ApMon(2, destList);
```

4. Adjust the settings for the ApMon object, if necessary (here we set the time interval for reloading the configuration page to 300 sec). This can also be done from the configuration file/webpage.

```
    apm -> setRecheckInterval(300);
```

5. Send datagrams; we used here two functions: one that includes a timestamp in the datagram and one that doesn't.

```
    for (i = 0; i > nDatagrams; i++) {
        /* generate a value for the parameter  */
        val += 0.05;
        if (val > 2)
            val = 0;
        /* use the logging facility */
        sprintf(logmsg, "Sending %lf for cpu_load\n", val);
        logger(INFO, logmsg);

        /* use the wrapper function with simplified syntax */

        try {
            apm -> sendParameter("TestCluster2_cpp", NULL, "my_parameter",
            XDR_REAL64, (char *)&val);

            /* send a datagram with timestamp */
            // as if we sent the datagram 5 hours ago
            timestamp = time(NULL) - (5 * 3600);
            apm -> sendTimedParameter(
                "TestClusterOld2_cpp",
                NULL,
                "my_parameter",
                XDR_REAL64,
                (char *)&val,
                timestamp
            ) ;
        } catch(runtime_error &e) {
            logger(WARNING, e.what());
        }

        sleep(2);
    } // for
} catch(runtime_error &e) {
    logger(WARNING, e.what());
}

return 0;
}
```

Chapter 4

ApMon Java User Guide

4.1 Installation

The ApMon archive contains the following files and folders:

- *apmon/* - package that contains the ApMon class and other helper classes:
  - *XDRDataOutput* and *XDROutputStream*, which are a part of a library for XDR encoding/decoding, provided under the LGPL license - see <`http://java.freehep.org`>
  - the *lisa_host* package, which contains classes from LISA - see http://monalisa.cacr.caltech.edu/dev_lisa.html <`http://monalisa.cacr.caltech.edu/dev_lisa.html`>.
- *lib/* - directory which will contain, after building, the libraries needed in order to use ApMon
- *examples/* examples for using the routines
- *lesser.txt* - the LGPL license for the XDR library
- *destinations.conf* - contains the IP addresses or DNS names of the destination hosts and the ports where the MonaLisa modules listen
- *build_apmon.sh, env_apmon* - for building on Linux systems
- *build_apmon.bat* - for building on Windows systems
- *README*
- *Doxyfile* - for generating Doxygen documentation

There is an additional directory, *ApMon_docs*, which contains the Doxygen documentation of the source files.

To build ApMon on Linux systems:

1. set the JAVA_HOME environment variable to the location where Java is installed
2. To build ApMon, cd to the ApMon directory and type:

   ```
   ./build_apmon.sh
   ```

   The ApMon jar file (*apmon.jar*) and a small Linux library (*libnativeapm.so*) are now available in the *lib/* directory. In order to use ApMon, the CLASSPATH must contain the path to *apmon.jar* and, optionally, the LD_LIBRARY_PATH must contain the path to *libnativeapm.so* (this library only provides one function, mygetpid(), which has the functionality of `getpid()`. You might want to use it for job monitoring, as in *Example_x1.java* and *Example_x2.java*). You can adjust the CLASSPATH and LD_LIBRARY_PATH manually or by sourcing the *env_apmon* script.

3. To build the ApMon examples, go to the examples/ directory and type:

   ```
   ./build_examples.sh
   ```

To build ApMon on Windows systems:

1. add the directory that contains the apmon package to the CLASSPATH
2. run build_apmon.bat
3. when running the examples, the directory *apmon\lisa_host\Windows* must be in the library path (the *system.dll* library from this directory will be used). This can be done by using the option `-Djava.library.path`:

   ```
   java -Djava.library.path=<path to apmon\lisa_host\Windows> exampleSend_1a
   ```

4.2 Using ApMon

4.2.1 ApMon Initialization

In the Java version, the ApMon features are available as methods of a class called ApMon. An ApMon object can be initialized with one of the constructors (see the API docs for more details):

`ApMon(String filename);`

- initializes ApMon from a configuration file whose name is given as parameter

`ApMon(Vector destList);`

- initializes ApMon from a vector which contain strings of the form "address:[port][password]" specifying destination hosts, and/or addresses of configuration webpages

`ApMon(Vector destAddresses, Vector destPorts, Vector destPasswds);`

- initializes ApMon from a list of destination hosts and the corresponding lists of ports and passwords

4.2.2 Sending Datagrams

There are two ways in which the user can send parameter values to MonALISA:

a a single parameter in a datagram

b multiple parameters in a datagram

For sending a datagram with a single parameter, the user should call the function `sendParameter()` which has several overloaded variants.

For sending multiple parameters in a datagram, the user should call the function `sendParameters()`, which receives as arguments arrays with the names and the values of the parameters to be sent.

Since version 2.0 there are two additional functions, `apMon_sendTimedParameter()` and `sendTimedParamet` which allow the user to specify a timestamp for the parameters.

IMPORTANT: When the ApMon object is no longer in use, the `stopIt()` method should be called in order to close the UDP socket used for sending the parameters.

4.2.3 Configuring ApMon with the aid of the API

The behaviour of ApMon can be configured not only from the configuration files or webpages, but also with the aid of the API. In order to enable the periodical reloading of the configuration files, the user should call `setConfCheck(true)`; the value of the time interval at which the recheck operatins are performed, can be set with the functions `setConfRecheck()` or `setRecheckInterval()`.

To enable/disable the automated job/system monitoring, and also to set the time intervals, the functions `setJobMonitoring()` and `setSysMonitoring()` can be used.

4.2.4 Automated Job Monitoring

To monitor jobs, you have to specify the PID of the parent process for the tree of processes that you want to monitor, the working directory, the cluster and the node names that will be registered in MonALISA (and also the job monitoring must be enabled). If work directory is "", no information will be retrieved about disk:

```
void addJobToMonitor(int pid, String workDir, String clusterName, String nodeName);
```

To stop monitoring a job, the `removeJobToMonitor(int pid)` should be called.

4.2.5 Logging

ApMon prints its messages to a file called apmon.log, with the aid of the `Logger` class from the Java API. The user may print its own messages with the logger (see the examples). The ApMon loglevels are FATAL (equivalent to Level.SEVERE), WARNING, INFO, FINE, DEBUG (equivalent to Level.FINEST). The ApMon loglevel can be set from the configuration file (by default it is INFO):

```
xApMon_loglevel = <level>
```

e.g.,

```
xApMon_loglevel = FINE
```

When setting the loglevel in the configuration file, you must use the ApMon level names rather than the Java names (so that the configuration file be compatible with the other ApMon versions).

The loglevel can also be set with the function `setLogLevel()` from the ApMon class.

4.2.6 Restrictions

The maximum size of a datagram is specified by the constant MAX_DGRAM_SIZE; by default it is 8192B and the user should not modify this value as some hosts may not support UDP datagrams larger than 8KB.

4.3 How to Write a Simple Java Program with ApMon

In this sect1 we show how the ApMon API can be used to write a simple program that sends monitoring datagrams. The source code for this short tutorial (slightly modified) can be found in the `Example_2.java`file under the *examples/* directory. The program generates some double values for a parameter called "my_cpu_load" and sends them to the MonALISA destinations. The number of iterations is given as a command line argument; in each iteration two datagrams are sent, one with timestamp and one without timestamp.

With this example program we'll illustrate the steps that should usually be taken to write a program with ApMon:

1. Import the ApMon package (and possibly other necessary packages):

```
import java.util.Vector;
import java.util.logging.Logger;

import apmon.*;
```

2. Initialize the variables we shall use..

```
public class Example_2 {
    private static Logger logger = Logger.getLogger("apmon");

    public static void main(String args[]) {
        Vector destList = new Vector();
        int nDatagrams = 20;
        ApMon apm = null;
        double val = 0;
        int i, timestamp;

        if (args.length == 1)
            nDatagrams = Integer.parseInt(args[0]);
```

3. Construct an ApMon object (in this example we used an intialization list containing the name of a destination host, ui.rogrid.pub.ro, and a webpage where other destination hosts and possibly ApMon options can be specified). The ApMon functions throw exceptions if errors appear, so it is recommended to place them in a try-catch block:

```
        destList.add(new String("ui.rogrid.pub.ro:8884 password"));
        destList.add(new String("http://lcfg.rogrid.pub.ro/˜corina/dest.conf"));

        try {
            apm = new ApMon(destList);
        } catch (Exception e) {
            logger.severe("Error initializing ApMon: " + e);
            System.exit(-1);
        }
```

4. Adjust the settings for the ApMon object, if necessary (here we set the time interval for reloading the configuration page to 300 sec, and we change the logging level to DEBUG). This can also be done from the configuration file.

```
        // set the time interval for periodically checking the
        // configuration URL
        apm.setRecheckInterval(300);
        // this way we can change the logging level
        apm.setLogLevel("DEBUG");
```

5. Send datagrams; we used here two functions: one that includes a timestamp in the datagram and one that doesn't.

```
        for (i = 0; i < nDatagrams - 1; i++) {
            val += 0.05;
            if (val > 2)
                val = 0;

            logger.info("Sending " + val + " for cpu_load");

            try {
                /* use the wrapper function with simplified syntax */
                apm.sendParameter("TestCluster2_java", null, "my_cpu_load",
                ApMon.XDR_REAL64, new Double(val));

                /* now send a datagram with timestamp (as if this was 5h ago) */
                long crtTime = System.currentTimeMillis();
                timestamp = (int)(crtTime / 1000 - (5 * 3600));
                apm.sendTimedParameter("TestClusterOld2_java", null, "my_cpu_load",
                ApMon.XDR_REAL64, new Double(val), timestamp);
            } catch(Exception e) {
                logger.warning("Send operation failed: " + e);
            }
            try {
            Thread.sleep(1000);
          } catch (Exception e) {}
        } // for
```

6. Stop ApMon:

```
        apm.stopIt();
    }
}
```

Chapter 5

ApMon Perl User Guide

5.1 Installation

The ApMon archive contains the following files in the ApMon module:

- *ApMon.pm* - main ApMon module. It can be instantiated by users to send data.
- *ApMon/Common.pm* - contains common functions for all other modules.
- *ApMon/XDRUtils.pm* - contains functions that encode different values in XDR format
- *ApMon/ProcInfo.pm* - procedures to monitor the system and a given application
- *ApMon/ConfigLoader.pm* - manages configuration retrieval from multiple places
- *ApMon/BgMonitor.pm* - handles the background monitoring of system and applications
- *README*
- *example/** - a set of examples with the usage of the ApMon module.
- *example/destinations.conf* - a sample destinations file, for url/file configuration
- *MAN* - a short description and API functions

To install this module type the following:

```
perl Makefile.PL
make
make test
make install
```

DEPENDENCIES:
This module requires these other modules and libraries:

```
Data::Dumper
LWP::UserAgent
Socket
Exporter
```

5.2 Using ApMon

5.2.1 ApMon Initialization

In the Perl version, the ApMon features are available as methods of a class called `ApMon`. The `ApMon` class will start a second process that will verify periodically the configuration files or webpages for any changes. The settings are communicated to the main procress through a local file, stored in temp. This is because there could be long waiting times for the downloading and the resolving of the destination hosts for the udp datagrams. This way, ApMon `sendParam*` functions will not block.
An ApMon object can be initialized in one of the following ways (see the manual for details):

- passing as parameters a list of locations from where the configuration will be loaded. These are strings that represent either a filename, or an URL address (if they start with `http://`).
- passing as parameter a reference to an array, containg directly the destinations (`host[:port][ pass]`). ApMon will send datagrams to all the valid given destinations (i.e. hosts that can be resolved), with default options. In this case, the child process that verifies configuration files for changes will be not be started and all function calls refering to this part will generate an error message.
- passing as parameter to the constructor a reference to a hash. In this case, the keys will be destinations and the corresponding values will be references to another hash in which the key is a configuration option and the value is the value of that option. Note that in this case, the options should not be preceded by `xApMon_` and options should be 0/1 instead of on/off as in the configuration file. In this case, the child process that verifies configuration files for changes will not be started and all function calls refering to this part will generate an error message. If system and job monitoring and general information are disabled, the child process that performs this monitoring will not be starting. Any attempts to setMonitorClusterNode or setJobPID will generate errors.
- passing no parameters. In this case, in order to initialize the destionations for the packets, you will have to use the `setDestinations()` function. This accepts the same parameters as the constructor, as described above.

5.2.2 Sending Datagrams

To send user parameters to MonALISA, you have the following set of functions:

```
sendParameters ( $clusterName, $nodeName, @params);
```

Use this to send a set of parameters to all given destinations. The default cluster an node names will be updated with the values given here.If afterwards you want to send more parameters, you can use the shorter version of this function, sendParams. The parameters to be sent can be eiter a list, or a reference to a list, a hash or a reference to a hash with pairs. This list should have an even length and should contain pairs like `(paramName, paramValue)`. `paramValue` can be a string, an integer or a float. Due to the way that Perl interprets functions parameters, you can put as many parameters in the function call as you want, not needing to create a list for this.

```
sendParams (@params);
```

Use this to send a set of parameters without specifying a cluster and a node name. In this case, the default values for cluster and node name will be used. See the sendParameters function for more details.

```
sendTimedParameters ($clusterName, $nodeName, $time, @params);
```

Use this instead of sendParameters to set the time for each packet that is sent. The time is in seconds from Epoch. If you use the other function, the time for these parameters will be sent by the MonALISA serice that receives them. Note that it is recommended to use the other version unless you really need to send the parameters with a different time, since the local time on the machine might not be synchronized to a time server. The MonALISA service sets the correct real time for the packets that it receives.

```
sendTimedParams ($time, @params);
```

This is the short version of the sendTimedParameters that uses the default cluster and node name to sent the parameters and allows you to specify a time (in seconds from Epoch) for each packet.
Please see the EXAMPLE file for examples of using these functions.

5.2.3 Configuring ApMon with the aid of the API

The behaviour of ApMon can be configured not only from the configuration files or webpages, but also with the aid of the API. In order to enable the periodical reloading of the configuration files / webpages, you should call `setConfRecheck($bool, [$interval]);` if you want, you can specify the time interval at which the recheck operatins are performed.

If you want to disable temporarily sending of background monitoring information, and to enable it afterwards, you can use:

```
enableBgMonitoring ($bool)
```

If you don't want to have the two background processes, you can turn them off whenever you want using the following function:

```
stopBgProcesses ();
```

In this case, configuration changes will not be notified anymore, and no background monitoring will be performed. To force a configuration check or send monitoring information about the system and/or jobs, you can use the following two functions:

```
refreshConfig ();
sendBgMonitoring ();
```

5.2.4 Automated Job Monitoring

To monitor jobs, you have to specify the PID of the parent process for the tree of processes that you want to monitor, the working directory, the cluster and the node names. If work directory is "", no information will be retrieved about disk:

```
addJobToMonitor ($pid, $workDir, $clusterName, $nodeName);
```

To stop monitoring a job, just call:

```
removeJobToMonitor ($pid);
```

5.2.5 Logging

To change the log-level of the ApMon module, you can either set it in the configuration file or use the following function:

```
setLogLevel(level);
```

with the following valid log-levels: "DEBUG", "NOTICE", "INFO", "WARNING", "ERROR", "FATAL"

From the configuration file the log-level can be set as follows:

```
xApMon_loglevel = <level>
```

e.g.,

```
xApMon_loglevel = FINE
```

5.2.6 Maximum number of messages

To set the maxim number of messages that can be sent to a MonALISA service, per second, you can use the follwing function:

```
setMaxMsgRate(rate);
```

By default, it is 50. This is a very large number, and the idea is to prevent errors from the user. One can easily put in a for loop, without any sleep, some sendParams calls that can generate a lot of unnecessary network load.

You can put this line in the config file:

```
xApMon_maxMsgRate = 30
```

The datagrams with general system information are only sent if system monitoring is enabled, at greater time intervals (one datagram with general system information for each 2 system monitoring datagrams).

5.2.7 Recreate the ApMon object

If you have to recreate the ApMon object over and over again, then you should use the following function when you fininshed using a instance of the ApMon:

```
free();
```

This will delete the temporary file and terminate the background processes. The UDP socket will not be closed, but if you will reuse ApMon afterwards, it will not open another socket.

5.2.8 Restrictions

The maximum size of a datagram is specified by the constant MAX_DGRAM_SIZE; by default it is 8192B and the user should not modify this value as some hosts may not support UDP datagrams larger than 8KB.

5.3 How to Write a Simple Perl Program with ApMon

In this sect1 we show how the ApMon API can be used to write a simple program that sends monitoring datagrams. The source code for this short tutorial (slightly modified) can be found in the simple_send.pl file under the *examples/* directory.

The program generates values for a few parameters and sends them to the MonALISA destinations; this action is repeated in 20 iterations.

With this example program we'll illustrate the steps that should usually be taken to write a program with ApMon:

1. Import the ApMon module (and possibly other necessary modules):

```
use strict;
use warnings;

use ApMon;
```

2. Initialize ApMon and possibly set some options (in this example, we disabled system monitoring). These options could have also been set from a configuration file, but here we don't have one.

```
# Initialize ApMon specifying that it should not send information about
# the system.
# Note that in this case the background monitoring process isn't stopped,
# in case you may want later to monitor a job.
my $apm = new ApMon({"pcardaab.cern.ch" => {"sys_monitoring" => 0,
                                              "general_info" => 0}});
```

3. Send the datagrams. We used here two functions: sendParameters, which specifies the cluster name and the node name (which will be cached in the ApMon object), and sendParams, which uses the names that were memorized at the call of the first function.

```
for my $i (1 .. 20) {
    # you can put as many pairs of parameter_name, parameter_value as you want
   # but be careful not to create packets longer than 8K.
   $apm->sendParameters("SimpleCluster", "SimpleNode",
                       "var_i", $i, "var_i^2", $i * $i);
   my $f = (20.0 / $i);
   # send in the same cluster and node as last time
   $apm->sendParams("var_f", $f, "5_times_f", 5 * $f, "i+f", $i + $f);
   sleep(1);
}
```

Chapter 6

ApMon Python User Guide

6.1 Installation

The ApMon archive contains the following files in the ApMon module:

- *apmon.py* - main ApMon module. It can be instantiated by users to send data.
- *ProcInfo.py* - procedures to monitor the system and a given application.
- *README*
- *example/*.py* - a set of examples with the usage of the ApMon module.
- *example/*.conf* - a set of sample destination files, for url/file configuration.
- *ApMon_doc/*.html* - HTML documentation

6.2 Using ApMon

6.2.1 ApMon Initialization

In the Python version, the ApMon features are available as methods of a class called `ApMon`. The *ApMon.pm* module defines the `ApMon` class that has to be instantiated in one of the following ways:

- passing as parameter a location from where the configuration will be loaded. This should be a string that represents either a filename, or an URL address (if it starts with "http://"). The `ApMon` class will start a second thread that will verify periodically the configuration file/webpage for any changes. This verification is done in a sepparate thread because there could be long waiting times for the downloading and the resolving of the destination hosts for the udp datagrams. This way, the `sendParam*` functions will not block. You can also pass an array of file/url strings in order to initialize ApMon.
- passing as parameter a tuple, containg as elements directly the destinations (`host[:port][ pass]`). ApMon will send datagrams to all valid given destinations (i.e.hosts that can be resolved), with default options.
- passing as parameter to the constructor a hash. In this case, the keys will be destinations and the corresponding values will be references to another hash in which the key is a configuration option and the value is the value of that option. Note that in this case, the options should not be preceded by xApMon_ and options should be True/False instead of on/off as in the configuration file.

6.2.2 Sending Datagrams

To send user parameters to MonALISA, you have the following set of functions:

```
sendParameters ( clusterName, nodeName, params);
```

Use this function to send a set of parameters to all given destinations. The default cluster an node names will be updated with the values given here. If afterwards you want to send more parameters, you can use the shorter version of this function, sendParams. The parameters to be sent can be eiter a list, or a hash. If list, it should have an even length and should contain pairs like (paramName, paramValue). paramValue can be a string, an integer or a float.

```
sendParams (params);
```

Use this to send a set of parameters without specifying a cluster and a node name. In this case, the default values for cluster and node name will be used. See the sendParameters function for more details.

```
sendTimedParameters (clusterName, nodeName, time, params);
```

Use this instead of sendParameters to set the time for each packet that is sent. The time is in seconds from Epoch. If you use the other function, the time for these parameters will be sent by the MonALISA service that receives them. Note that it is recommended to use the other version unless you really need to send the parameters with a different time, since the local time on the machine might not be synchronized to a time server. The MonALISA service sets the correct real time for the packets that it receives.

Please see the examples/*.py files for examples of using these functions.

6.2.3 Configuring ApMon with the aid of the API

The behaviour of ApMon can be configured not only from the configuration files or webpages, but also with the aid of the API.

If you want to disable temporarily sending of background monitoring information, and to enable it afterwards, you can use:

```
enableBgMonitoring (bool)
```

To force sending monitoring information with background monitoring disabled, you can use the following function:

```
sendBgMonitoring ();
```

6.2.4 Automated Job Monitoring

To monitor jobs, you have to specify the PID of the parent process for the tree of processes that you want to monitor, the working directory, the cluster and the node names. If work directory is ″″, no information will be retrieved about disk:

```
addJobToMonitor (pid, workDir, clusterName, nodeName);
```

To stop monitoring a job, just call:

```
removeJobToMonitor (pid);
```

6.2.5 Logging

To change the log-level of the ApMon module, you can either set it in the configuration file or use the following function:

```
setLogLevel(level);
```

with the following valid log-levels: ″DEBUG″, ″NOTICE″, ″INFO″, ″WARNING″, ″ERROR″, ″FATAL″

From the configuration file the log-level can be set as follows:

```
xApMon_loglevel = <level>
```

e.g.,

```
xApMon_loglevel = FINE
```

6.2.6 Maximum number of messages

To set the maxim number of messages that can be sent to a MonALISA service, per second, you can use the follwing function:

```
setMaxMsgRate(rate);
```

By default, it is 50. This is a very large number, and the idea is to prevent errors from the user. One can easily put in a for loop, without any sleep, some sendParams calls that can generate a lot of unnecessary network load.

You can put this line in the config file:

```
xApMon_maxMsgRate = 30
```

The datagrams with general system information are only sent if system monitoring is enabled, at greater time intervals (one datagram with general system information for each 2 system monitoring datagrams).

6.2.7 Recreate the ApMon object

If you have to recreate the ApMon object over and over again, then you should use the following function when you fininshed using a instance of the ApMon:

```
free();
```

This will delete the temporary file and terminate the background processes. The UDP socket will not be closed, but if you will reuse ApMon afterwards, it will not open another socket.

6.2.8 Restrictions

The maximum size of a datagram is specified by the constant MAX_DGRAM_SIZE; by default it is 8192B and the user should not modify this value as some hosts may not support UDP datagrams larger than 8KB.

6.3 How to Write a Simple Python Program with ApMon

In this sect1 we show how the ApMon API can be used to write a simple program that sends monitoring datagrams. The source code for this short tutorial (slightly modified) can be found in the *simple_send.py* file under the *examples/* directory.

The program generates values for a few parameters and sends them to the MonALISA destinations specified in the 'dest_2.conf' file; this action is repeated in 20 iterations.

With this example program we'll illustrate the steps that should usually be taken to write a program with ApMon:

1. Import the ApMon module (and possibly other necessary modules):

```
import apmon
import time
```

2. Initialize ApMon and possibly set some options (in this example, we disabled the background job/system monitoring and the periodical reloading of the configuration file, and also set the loglevel). These options could have also been set from a configuration file.

```
# Initialize ApMon specifying that it should not send information about
# the system.
# Note that in this case the background monitoring process isn’t stopped,
# in case you may want later to monitor a job.
apm = apmon.ApMon(’dest_2.conf’)
apm.setLogLevel("INFO");
apm.confCheck = False
apm.enableBgMonitoring(False)
```

3. Send the datagrams. We used here two functions: sendParameters, which specifies the cluster name and the node name (which will be cached in the ApMon object), and sendParams, which uses the names that were memorized at the call of the first function.

```
for i in range(1,20):
    # you can put as many pairs of parameter_name, parameter_value as you want
    # but be careful not to create packets longer than 8K.
    apm.sendParameters("SimpleCluster", "SimpleNode",
                       {’var_i’: i, ’ar_iˆ2’: i*i})
    f = 20.0 / i
    # send in the same cluster and node as last time
    apm.sendParams({’var_f’: f, ’5_times_f’: 5 * f, ’i+f’: i + f})
    print "simple_send-ing for i=",i
    time.sleep(1)
```