Monitoring daemon

Version: 2025-05-21

Table of Contents

1 Introduction

Monitoring daemon is a software daemon allowing to monitor state of other Amarisoft software components:

- eNB
- MME
- IMS
- UE
- MBMSGW

It allows to log events such as component starting, stopping or related errors.
It can also trigger alarms sent by email.
A daemon instance can be set as a proxy to gather information from other remote daemon on different machines.

2 Requirements

2.1 Software requirements

- A 64 bit Linux distribution. Fedora 39 is the officially supported distribution. The following distributions are known as compatible:
  - Fedora 22 to 39
  - Cent OS 7
  - Ubuntu 14 to 22

  Your system requires at least GLIBC 2.17.

3 Installation

LTEMONITOR can be run directly from the directory when it was unpacked.
However, for convenience, we recommend to install it from the `install.sh` script of your Amarisoft release tarball.

3.1 Linux setup

3.1.1 Packages

LTEMONITOR requires `nodejs` to run.
`ssmtp` may also be required to send alarms through emails.
Note that those package are automatically installed by `install.sh` script.

3.2 License key installation

LTEMONITOR does not require any license key to run.

4 Events

Purpose of LTEMONITOR is to monitor events coming from software components.
Events are dumped into log file and may generate alarms.
An event is defined by the following properties:

- time Timestamp of the event.
- hostname Name of the machine the event has occurred on.
- level Criticity of the event.
- component Software component.
- section Section within software component.
- title Title of the event.
- message Message of the event.

4.1 Level

Event level can be:

- ERROR
- WARN
- INFO
- DEBUG

4.2 State event

Those events apply to all software components.

Components	**Sections**	**Titles**	**Level**
All	STATE	started stopped	`INFO` or `WARN` if component was previously in error state `INFO`

4.3 Network link events

Network link of software component generate event when their state is changing.
The `section` is the link name and the `title` can be:

- connected
- disconnected

In case of connection error, `disconnected` event will be raised with `WARN` level.
When the connection becomes up again, `connected` event will be raise with `WARN` level.
For other cases, the level is always `INFO`.

`section` will identify link type and depends on `component` as below:

Components	**Sections**	**Titles**
eNB	S1, NG	connected disconnected

MME	Rx, Cx, S6, S13, SGs, N12	connected disconnected
IMS	Rx, Cx	connected disconnected
PROXY	proxy-link	connected disconnected

4.4 Error events

The following events always have a level set to `ERROR`.

Components	**Section**	**Title**	**Definition**
All	AUTH	failure	Occurs when monitor fails to connect to component
All	CONFIG	Syntax error Missing internal error	 Config file syntax error Config file is missing Internal error
All	INIT	Unexpected termination License error Config error	Software component has accidentaly stopped (crash, signal...) Component can't start because of license Component can't start because of invalid configuration file
All	RUNTIME	Unexpected termination License error	Software component has accidentaly stopped (crash, signal...) Component stops working because of license
eNB UE	INIT	RF frontend	Radio frontend initialization error
Monitor	SYSTEM	High disk usage	System disk is being full

5 Email configuration

LTEMONITOR uses ssmtp to end emails.
Thus, you need a file to configure your SMTP server.
The syntax of the file can be found on `https://wiki.archlinux.org/index.php/SSMTP` or by looking at /etc/ssmtp/ssmtp.conf.

Once you have created your configuration file, you may try it with the `test-email.sh` script you will find under LTEMONITOR directory.

Example:

```
./test-email.sh send@amarisoft.com receiver@amarisoft.com ssmtp.conf
[<-] 220 xxxxxxxxxxxxxxxxxxxx ESMTP Postfix
[->] EHLO amarisoft.com
[<-] 250 DSN
[->] AUTH LOGIN
[<-] 334 xxxxxxxxxxxx
[->] xxxxxxxxxxxxxxxxxxxxxxxx
[<-] 334 xxxxxxxxxxxx
[<-] 235 2.7.0 Authentication successful
[->] MAIL FROM:<sender@amarisoft.com>
[<-] 250 2.1.0 Ok
[->] RCPT TO:<receiver@amarisoft.com>
[<-] 250 2.1.5 Ok
[->] DATA
[<-] 354 End data with <CR><LF>.<CR><LF>
[->] Received: by amarisoft.com (sSMTP sendmail emulation); Thu, 04 Jun 2020 18:32:09 +0200
[->] Date: Thu, 04 Jun 2020 18:32:09 +0200
[->] From: receiver@amarisoft.com
[->] To: sender@amarisoft.com
[->] Subject: Amarisoft monitor e-mail test
[->]
[->] Sent from xxxxxxxxxxxxxxx at Thu 04 Jun 2020 06:32:09 PM CEST
[->]
[->] .
[<-] 250 2.0.0 Ok: queued as 30B2D40008
[->] QUIT
[<-] 221 2.0.0 Bye
```

You can also test emails sent by LTEMONITOR by running it in command line this way:

```
./ltemonitor.js config/monitor.cfg --test-email 'emailID'
```

This will send an email from template according to your configuration file.
Note that `lte` service should be stopped.

6 Statistics

Software component are able to provide statistics via `stats` remote API.
LTEMONITOR is able to aggregate those statistics over time and provide perdiodical reports.

See [Statistics configuration], page 15.

Statistics are stored in a directory defined by configuration.
A subdirectory named by hostname will contain the information for the local machine and each machine connected via proxy.

Inside each host subdirectory, LTEMONITOR will store files with the following pattern:

```
YYYYMMDD-HH:MM:SS-<COMPONENT ID>.<STATS TYPE>
```

Statistics type can be:

- stats Global statistics
- cdr Common usage statistics

Each file is in JSON format representing an object.
All statistics files have the following common members:

`type`	String. `stats` or `cdr`.
`info`	Object with following members:

`version`	Integer. Statistic version
`id`	String. Component ID.
`name`	String. Component name.
`start`	Number. Start time in seconds (Since Jan 1st 1970).
`end`	Number. End time in seconds (Since Jan 1st 1970).
`hostname`	String. Component hostname
`custom_fields`	Optional array. Custom fields from host monitor custom_fields configuration (See [custom_fields], page 14). Each element of the array is an object representing a custom field with its `id`, `title` and `value`.

6.1 Stats

Global statistics depend on software components and have following members

`counters`	Protocol message counters. Check definition in `stats` remote API message (Remote API/Common message section) of the associated software component documentation.
`pdn_list`	PDN statistics for `MME` component. Check definition in `stats` remote API message (Remote API/Common message section) of `MME` component documentation.
`cells`	Per cells statistics for `ENB` component. Check definition in `stats` remote API message (Remote API/Common message section) of `ENB` component documentation.

6.2 cdr

CDR data can have following members:

`bearers` For `MME` component.
Array of bearers with following properties:

`imsi`	String. Associated UE IMSI.
`imei`	String. Associated UE IMEI.
`apn`	String. Access point name
`pdn_type`	String. `ipv4`, `ipv6`, `ipv4v6`, `non-ip` or `unstructured`.
`ipv4_addr`	String. IPv4 address.
`ipv6_prefix`	String: IPv6 address prefix
`dl_bytes`	Integer. Number of downlink bytes gone through this bearer.
`ul_bytes`	Integer. Number of uplink bytes gone through this bearer.
`duration`	Number. Bearer lifetime in seconds.
`start_date`	Integer. Bearer establishment date in seconds since Jan 1st 1970.

`sms` For `IMS` component.
Array of SMS event with following properties:

`sender`	String. SMS sender IMPU.
`destination`	String. SMS destination IMPU.
`type`	String. `MO` or `MT`.

`calls` For `IMS` component.
Array of IMS calls with following properties:

`type`	String. `MO` or `MT`.
`from`	String. SMS sender IMPU.
`to`	String. SMS destination IMPU.
`date`	Integer. Call establishment date in seconds since Jan 1st 1970.
`duration`	Number. Call duration in seconds.
`send_size`	Integer. Number of media data sent in bytes.
`recv_size`	Integer. Number of media data received in bytes.

7 Configuration reference

7.1 Configuration file syntax

The main configuration file uses a syntax very similar to the Javascript Object Notation (JSON) with few extensions.

1. Supported types:
   - Numbers (64 bit floating point). Notation: `13.4`
   - Complex numbers. Notation: `1.2+3*I`
   - Strings. Notation: `"string"`
   - Booleans. Notation: `true` or `false`.
   - Objects. Notation: `{ field1: value1, field2: value2, .... }`
   - Arrays. Notation: `[ value1, value2, .... ]`
2. The basic operations `+`, `-`, `*` and `/` are supported with numbers and complex numbers. `+` also concatenates strings. The operators `!`, `||`, `&&`, `==`, `!=`, `<`, `<=`, `>=`, `>` are supported too.
3. The numbers `0` and `1` are accepted as synonyms for the boolean values `false` and `true`.
4. {} at top level are optional.
5. `"` for property names are optional, unless the name starts with a number.
6. Properties can be duplicated.
   If properties are duplicated, they will be merged following [JSON merge rules], page 10, with overriding occuring in reading direction (last overrides previous).
   Ex:
   ```
   {
       value: "foo",
       value: "bar",
       sub: {
           value: "foo"
       },
       sub: {
           value: "bar"
       }
   }
   ```
   Will be equivalent to:
   ```
   {
       value: "bar",
       sub: {
           value: "bar"
       }
   }
   ```
7. Files can be included using *include* keyword (must not be quoted) followed by a string (without :) representing the file to include (path is relative to current file) and terminating by a comma.
   Arrays can't be included.
   Merge will be done as for duplicate properties.
   If *file1.cfg* is:
   ```
       value: "foo",
       include "file2.cfg",
       foo: "foo"
   ```

And *file2.cfg* is:

```
value: "bar",
foo: "bar"
```

Final config will be:

```
{
    value: "bar",
    foo: "foo"
}
```

8. A C like preprocessor is supported. The following preprocessor commands are available:

   `#define` *var expr*
   : Define a new variable with value *expr*. *expr* must be a valid JSON expression. Note that unlike the standard C preprocessor, *expr* is evaluated by the preprocessor.

   `#undef` *var*
   : Undefine the variable *var*.

   `#include` *expr*
   : Include the file whose filename is the evaluation of the string expression *expr*.

   `#if` *expr* Consider the following text if *expr* is true.

   `#else` Alternative of `#if` block.

   `#elif` Composition of `#else` and `#if`.

   `#endif` End of `#if` block.

   `#ifdef` *var*
   : Shortcut for `#if defined(var)`

   `#ifndef` *var*
   : Shortcut for `#if !defined(var)`

   In the JSON source, every occurrence of a defined preprocessor variable is replaced by its value.

9. Backquote strings: JSON expression can be inserted in backquote delimited strings with the `${expr}` syntax. Example: `‘abc${1+2}d‘` is evaluated as the string `"abc3d"`. Preprocessor variables can be used inside the expression. Backquote strings may span several lines.

7.1.1 JSON merge rules

Merge overriding direction depends on context, i.e source may override destination or the opposite.
JSON merge is recursive for Objects and Arrays.

Example, merging

```
{
    foo: { value: "bar" },
    same: "one",
    one: 1
}
```

with

```
{
    foo: { value: "none", second: true },
    same: "two",
    two: 1
}
```

Will become:

```
{
    foo: { value: "bar", second: true },
    same: "one",
    one: 1
    two: 1
}
```

assuming first object overrides second one.

In case of Array merging, the final array length will be the maximum length of all merged arrays.
For each element of the final array, merge will be done considering defined elements only.
Ex:

```
{
    array: [0, 1, 2, { foo: "bar" } ],
    array: [3, 4],
    array: [5, 6, 7, { bar: "foo" }, 8 ]
}
```

Will be merged to:

```
{
    array: [5, 6, 7, { foo: "bar", bar: "foo" }, 8 ],
}
```

7.2 Properties

`log_filename`
: String. Set the log filename. If no leading `/`, it is relative to the configuration file path. See [Log file format], page 25.

`log_options`
: String. Set the logging options as a comma separated list of assignments.

- *layer*.level=*verbosity*. For each layer, the log verbosity can be set to `none`, `error`, `info` or `debug`. In debug level, the content of the transmitted data is logged.
- *layer*.max_size=*n*. When dumping data content, at most `n` bytes are shown in hexa. For ASN.1, NAS or Diameter content, show the full content of the message if `n > 0`.
- *layer*.payload=[0|1]. Dump ASN.1, NAS, SGsAP or Diameter payload in hexadecimal.
- *layer*.key=[0|1]. Dump security keys (NAS and RRC layers).
- *layer*.crypto=[0|1]. Dump plain and ciphered data (NAS and PCDP layers).
- time=[sec|short|full]. Display the time as seconds, time only or full date and time (default = time only).
- time.us=[0|1]. Dump time with microseconds precision.
- file=cut. Close current file log and open a new one.

- file.rotate=now. Move and rename to the same directory or to the directory pointed by `file.path` and open a new log file (Headers are kept).
- file.rotate=*size*. Every time log file size reaches *size* bytes, move and rename to the same directory or to the directory pointed by `file.path`, and open a new log file (Headers are kept).
  Size is an integer and can be followed by K, M or G.
- file.rotate=#*count*. Everytime number of logs in log file reaches *count*, move and rename to the same directory or to the directory pointed by `file.path`, and open a new log file (Headers are kept).
  Size is an integer and can be followed by K, M or G.
- file.path=*path*. When log rotation is enabled (`file.rotate` set), rename and move current log to this path instead of initial log path.
- append=[0|1]. (default=0). If 0, truncate the log file when opening it. Otherwise, append to it.

Available layers are: `MON`, `EVENT` or `<component name>`

`log_sync` Optional boolean (default = false). If true, logs will be synchronously dumped to file.
Warning, this may lead to performances decrease.

`com_addr` Optional string. Address of the WebSocket server remote API. See [Remote API], page 17.
If set, the WebSocket server for remote API will be enabled and bound to this address.
Default port is 9007.
Setting IP address to [::] will make remote API reachable through all network interfaces.

`com_name` Optional string. Sets server name. MONITOR by default

`com_log_lock`
Optional boolean (default is false). If *true*, logs configuration can't be changed via `config_set` remote API.

`com_log_us`
Optional boolean (default is false). If *true*, logs sent by `log_get` remote API response will have a `timestamp_us` parameters instead of `timestamp`

`com_auth` Optional object. If set, remote API access will require authentication.
Authentication mechanism is describe in [Remote API Startup], page 19, section.

`passfile` Optional string. Defines filename where password is stored (plaintext).
If not set, `password` must be set

`password` Optional string. Defines password.
If not set, `passfile` must be set.

`unsecure` Optional boolean (default false). If set, allow password to be sent plaintext.
NB: you should set it to true if you access it from a Web Browser (Ex: Amarisoft GUI) without SSL (https) as your Web Browser may prevent secure access to work.

`com_log_count`
Optional number (Default = 8192). Defines number of logs to keep in memory before dropping them.
Must be between 4096 and 2097152).

`proxy` Optional object. If set, events and alarms will be forwarded to other instance of LTEMONITOR on its remote API interface (`com_addr`), See [Proxy], page 14.

`stats` Optional object. If set, statistics will be aggregated to generate periodical reports. See [Statistics], page 6, and See [Statistics configuration], page 15,

`backup.` Optional object. If set, configuration directories of each software component will be backed-up. See [Backup configuration], page 17,

`emails` Optional array of object. Each object represent an email configuration that may be used to send alarms.
Each email has the following properties:

- `id` String. Alarm ID. If not set, this email configuration will be the default one, else alarms with same ID will use this configuration.
  Note that `id` must be unique and only one default configuration is allowed.
- `from` String. Sender email address.
- `to` String. Email address to send email alarm to.
- `smtp` String. SSMTP configuration file, See [SSMTP], page 5.
- `template` String. Email template file.
  Emails will be generated using this template file (See [Email template], page 14).
- `aggregation`
  Optional object. If present, defines the aggregation method to avoid sending too many emails for an alarm.
  The object is defined this way:
  - `delay` Number. Time in seconds for alarm aggregation aggregation.
  - `count` Integer. Minimum amount of generated alarms to trigger an email sending.

  When an alarm is triggered, a first email is sent. If same alarm is generated within the `delay` period, no email will be sent. When the `delay` has expired, an email with aggregated information will be sent if at least `count` alarms have been triggered.
  In that case, the `delay` period will be restarted, else, the aggregation mechanism is disabled until a new alarm happens.
  Alarms will be aggregated if they have same `LEVEL`, `HOST`, `COMP` and `TITLE`.

`alarms` Array of object. Each item will represent an alarm trigger (See [Alarms], page 15). Each event will be compared to alarms filter and if it matches one of them, alarm will be generated.
If multiple alarms definition matches, multiple alarms may be raised.

`custom_definition`
Optional array of object. Each item will represent a custom column displayed within monitor tab of web interface. Each column has the following parameters:

- title Title of the column

- id ID of the field
- stats Optional boolean (default: true). If set to false, custom field won't be dumped inside stats files.

`custom_fields`
: Optional object. Each property name represents the ID of the field and each property value represents its value to be displayd in web interface.

`system`
: Optional object. If set, LTEMONITOR will monitor system state.
Properties are the following:

  `poll_delay`
  : Optional number (default = 60). Delay in seconds for polling system state and generating alarms.

  `disk_usage`
  : Optional number (default = 95). Percentage of disk usage to consider disk as full and generating alarm.

7.2.1 Proxy

When a proxy is set, events will be sent to the designated LTEMONITOR.
Alarms will be triggered by local `alarms` configuration but email sending will be up to remote proxy.

State of the component will also be forwarded to remote proxy.

If remote proxy is not reachable, local LTEMONITOR may store events on local storage until connection is successful.

`addr`
: String. IP address of the remote LTEMONITOR instance.

`store`
: String. If set, represents directory name where not forwarded events are stored.
Events are permanently stored and will resist upon service restart.
If value is not an absolute path, it will be relative to configuration file location.

`timeout`
: Number. Maximum lifetime in seconds of events in store.
Events will be removed if proxy has not been reached for longer than this value.

`keepalive`
: Optional number (default = 30). Connection timeout in seconds.

`bulk`
: Optional integer (default = 20). Represents maximum number of events sent at once to proxy.

7.2.2 Email template

Emails template is a file from which emails will be generated.
It can include comments delimited by # character.

It has to follow rfc2822 (`https://tools.ietf.org/html/rfc2822`).

Which means that it must starts with headers terminated by an empty line and followed by email body.

Note that at least `Subject` header as `From` and `To` headers are generated by LTEMONITOR.

You may take a look at `alarm.tpl` example provided in `config` subdirectory.

Inside email template, you can use variables that will be replaced during email generation by the information related to event.
Here is the list of variables:

- <HOST> Hostname of the system where the event has occurred
- <LEVEL> Level of the event
- <COMPONENT> Component name
- <SECTION> Section of the event
- <TITLE> Title of the event
- <ALARM> Alarm ID associated to event
- <DATE> Date of the event
- <VERSION> Software version of the component
- <MESSAGE> Message of the event
- <COUNT> Number of associated alarms. 1 or greater in case of aggregation (See [alarms aggregation], page 13)

7.2.3 Alarms

Each alarm has following properties:

`id` String. Alarm ID. Used to match email configuration.

`priority` Optional number (default = 0). When an event matches multiple alarms filters, only the alarm with the highest priority will be triggered.
Multiple alarms can have the same priority.

`filters` Array of object. To trigger an alarm, event must match one of the filters.
A filter is an object where each property/value couple represents an event property and the pattern to apply on its value.
Property names are:

- level
- component
- section
- title

Value of each property is a string representing a regular expression (JavaScript convention).
All properties must match to trigger the alarm.
If no property is set (Filter is an empty object), filter will always match.

Examples:

```
{
  level: 'ERROR|WARN'
}
```

Here, alarms will be triggered for all events that has a level of ERROR or WARN

```
{
  section: 'Rx|Cx',
  title: '.*connected'
}
```

Here, alarms will be triggered on events `connected` and `disconnected` generated by `Rx` or `Cx` section.

7.2.4 Statistics configuration

Stats configuration is defined as followed:

`time` String. CRON style string defining when to generate report.
The format is "<week day>:<month>:<month day>:<hour>:<minute>:<seconds>" where each element can be:

- * meaning report will be generate on each value.
- <n> where <n> is a number defining the report generation time.
- <a>-<b> where <a> and <b> are numbers defining a range when the report generation will occur
- <n1>,<n2>,...,<ni> where each <nx> is one of the other definition above.

If less than 6 elements are defined, `time` is completed (on right side) by * for each missing elements (Ex: "2:0:0" means "*:*:*:2:0:0:").

Examples:

```
"22:0:0"
1 report will be generated every day at 10pm.

"4,22:0:0"
2 reports will be generated every day at 4am and 10pm.

"0:*/5:0"
A report will be generated every 5 minutes.

"*:5-10,15:0"
6 reports will be generated every hour at minutes 5, 6, 7, 8, 9, 10 and 15.


"1:*:*:14:0:0"
One report will be generated each monday at 2pm.
```

`utc` Optional boolean (default = true). Defines whether `time` parameter is considered as UTC time or local time.

`store` String. If set, represents directory name where report will be stored in JSON format. If value is not an absolute path, it will be relative to configuration file location. This parameter is mandatory if `proxy` is not set.
If proxy is set, the store won't be used and reports will be forwarded to remote LTEMONITOR.
See [Statistics], page 6, for details about data format

`timeout` Number. Mandatory if `proxy` is not set. Lifetime in seconds of generated report files.

`indent` Optional number (default = 0). If set, statistics dumped to store will be indented.

`group` Optional string. If set, group owner of stats file will be set to this value.

`permission`
Optional number (default = 0640). Stats file unix style permission.

`enabled` Optional boolean (default = true). Enable or disable statistics gathering on local machine. It does not prevent proxy client/server communication.

`comp_poll_delay`
Optional number (default = 5). Poll interval in seconds of stats for each component.

`sub_sampling`
Optional number. If set, some statistics will have a shorter sampling time and will be aggregate inside stat files with an array. The value represents the stats sampling time in seconds.

7.2.5 Backup configuration

Backup mechanism allows to backup configuration of each software component, including remote one connected through proxy.

`time` See [Statistics configuration], page 15,

`utc` See [Statistics configuration], page 15,

`store` String. If set, represents directory name where report will be stored in JSON format. If value is not an absolute path, it will be relative to configuration file location.

`timeout` See [Statistics configuration], page 15,

`group` Optional string. If set, group owner of stats file will be set to this value.

`permission`
See [Statistics configuration], page 15,

8 Remote API

You can access LTEMONITOR via a remote API.
Protocol used is WebSocket as defined in RFC 6455 (`https://tools.ietf.org/html/rfc6455`).

Note that Origin header is mandatory for the server to accept connections.
This behavior is determined by the use of `nopoll` library.
Any value will be accepted.

To learn how to use it, you can refer to our the following tutorial (`https://tech-academy.amarisoft.com/RemoteAPI.html`).

8.1 Messages

Messages exchanged between client and LTEMONITOR server are in strict JSON format.

Each message is represented by an object. Multiple message can be sent to server using an array of message objects.

Time and delay values are floating number in seconds.

There are 3 types of messages:

- Request

  Message sent by client.
  Common definition:

  `message` String. Represent type of message. This parameter is mandatory and depending on its value, other parameters will apply.

  `message_id`
  Optional any type. If set, response sent by the server to this message will have same message_id. This is used to identify response as WebSocket does not provide such a concept.

  `start_time`
  Optional float. Represent the delay before executing the message.
  If not set, the message is executed when received.

  `absolute_time`
  Optional boolean (default = false). If set, `start_time` is interpreted as absolute.
  You can get current clock of system using `time` member of any response.

  `standalone`
  Optional boolean (default = false). If set, message will survive WebSocket disconnection, else, if socket is disconnected before end of processing, the message will be cancelled.

  `loop_count`
  Optional integer (default = 0, max = 1000000). If set, message will be repeated `loop_count` time(s) after `loop_delay` (From message beginning of event).
  Response will have a `loop_index` to indicate iteration number.

`loop_delay`
: Optional number (min = 0.1, max = 86400). Delay in seconds to repeat message from its `start_time`. Mandatory when `loop_count` is set > 0.

- Response

  Message sent by server after any request message as been processed.
  Common definition:

  `message` String. Same as request.

  `message_id`
  Optional any type. Same as in request.

  `time` Number representing time in seconds since start of the process.
  Usefull to send command with absolute time.

  `utc` Number representing UTC seconds.

- Events

  Message sent by server on its own initiative.
  Common definition:

  `message` String. Event name.

  `time` Number representing time in seconds.
  Usefull to send command with absolute time.

8.2 Startup

When WebSocket connections is setup, LTEMONITOR will send a first message with name set to `com_name` and type set to MONITOR.

If authentication is not set, message will be `ready`:

```
{
    "message": "ready",
    "type": "MONITOR",
    "name": <com_name>,
    "version": <software version>,
    "product": <Amarisoft product name (optional)>
}
```

If authentication is set, message will be `authenticate` :

```
{
    "message": "authenticate",
    "type": "MONITOR",
    "name": <com_name>,
    "challenge": <random challenge>
}
```

To authenticate, the client must answer with a `authenticate` message and a `res` parameter where:

```
res = HMAC-SHA256( "<type>:<password>:<name>", "<challenge>" )
```

`res` is a string and HMAC-SHA256 refers to the standard algorithm (`https://en.wikipedia.org/wiki/HMAC`)

If the authentication succeeds, the response will have a `ready` field set to `true`.

```
{
    "message": "authenticate",
    "message_id": <message id>,
    "ready": true
}
```

If authentication fails, the response will have an `error` field and will provide a new challenge.

```
    {
        "message": "authenticate",
        "message_id": <message id>,
        "error": <error message>,
        "type": "MONITOR",
        "name: <name>,
        "challenge": <new random challenge>
    }
```

If any other message is sent before authentication succeeds, the error `"Authentication not done"` will be sent as a response.

8.3 Errors

If a message produces an error, response will have an error string field representing the error.

8.4 Sample nodejs program

You will find in this documentation a sample program: `ws.js`.
It is located in `doc` subdirectory.
This is a nodejs program that allow to send message to LTEMONITOR.
It requires nodejs to be installed:

```
    dnf install nodejs npm
    npm install nodejs-websocket
```

Use relevant package manager instead of NPM depending on your Linux distribution.

Then simply start it with server name and message you want to send:

```
    ./ws.js 127.0.0.1:9007 '{"message": "config_get"}'
```

8.5 Common messages

`config_get`
Retrieve current config.

Response definition:

`type` Always "MONITOR"

`name` String representing server name.

`logs` Object representing log configuration.
With following elements:

`layers` Object. Each member of the object represent a log layer configuration:

`layer name`
Object. The member name represent log layer name and parameters are:

`level` See [log_options], page 11,

`max_size`	See [log_options], page 11,
`key`	See [log_options], page 11,
`crypto`	See [log_options], page 11,
`payload`	See [log_options], page 11,

`count`	Number. Number of bufferizer logs.
`rotate`	Optional number. Max log file size before rotation.
`rotate_count`	Optional number. Max log count before rotation.
`path`	Optional string. Log rotation path.
`bcch`	Boolean. True if BCCH dump is enabled (eNB only).
`mib`	Boolean. True if MIB dump is enabled (eNB only).

`locked` Optional boolean. If `true`, logs configuration can't be changed with `config_set` API.

`config_set`

Change current config.
Each member is optional.
Message definition:

`logs` Optional object. Represent logs configuration. Same structure as config_get (See [config_get logs member], page 20).
All elements are optional.
Layer name can be set to `all` to set same configuration for all layers.
If set and logs are locked, response will have `logs` property set to `locked`.

`log_get` Get logs.
This API has a per connection behavior. This means that the response will depend on previous calls to this API within the same WebSocket connection.
In practice, logs that have been provided in a response won't be part of subsequent request unless connection is reestablished. To keep on receiving logs, client should send a new `log_get` request as soon as the previous response has been received.
If a request is sent before previous request has been replied, previous request will be replied right now without considering specific min/max/timeout conditions.

Message definition:

`min`	Optional number (default = 1). Minimum amount of logs to retrieve. Response won't be sent until this limit is reached (Unless timeout occurs).
`max`	Optional number (default = 4096). Maximum logs sent in a response.
`timeout`	Optional number (default = 1). If at least 1 log is available and no more logs have been generated for this time, response will be sent.
`allow_empty`	Optional boolean (default = false). If set, response will be sent after timeout, event if no logs are available.
`rnti`	Optional number. If set, send only logs matching rnti.
`ue_id`	Optional number. If set, send only logs with matching ue_id.

`layers` Optional Object. Each member name represents a log layer and values must be string representing maximum level. See [log_options], page 11.
If *layers* is not set, all layers level will be set to *debug*, else it will be set to *none*.
Note also the logs is also limited by general log level. See [log_options], page 11.

`short` Optional boolean (default = false). If set, only first line of logs will be dumped.

`headers` Optional boolean. If set, send log file headers.

`start_timestamp`
Optional number. Is set, filter logs older than this value in milliseconds.

`end_timestamp`
Optional number. Is set, filter logs more recent than this value in milliseconds.

`max_size` Optional number (default = 1048576, i.e. 1MB). Maximum size in bytes of the generated JSON message. If the response exceeds this size, the sending of logs will be forced independently from other parameters.

Response definition:

`logs` Array. List of logs. Each item is a an object with following members:

- `data` Array. Each item is a string representing a line of log.
- `timestamp`
  Number. Milliseconds since January 1st 1970. Not present if `com_log_us` is set in configuration.
- `timestamp_us`
  Number. Microseconds since January 1st 1970. Only present if `com_log_us` is set in configuration.
- `layer` String. Log layer.
- `level` String. Log level: *error*, *warn*, *info* or *debug*.
- `dir` Optional string. Log direction: *UL*, *DL*, *FROM* or *TO*.
- `ue_id` Optional number. UE_ID.
- `cell` Optional number (only for PHY layer logs). Cell ID.
- `rnti` Optional number (only for PHY layer logs). RNTI.
- `frame` Optional number (only for PHY layer logs). Frame number (Subframe is decimal part).
- `channel` Optional string (only for PHY layer logs). Channel name.
- `src` String. Server name.
- `idx` Integer. Log index.
- `headers` Optional array. Array of strings.

`discontinuity`
: Optional number. If set, this means some logs have been discarded due to log buffer overflow.

`microseconds`
: Optional boolean. Present and set to true if `com_log_us` is set in configuration file.

`log_set`
: Add log.
Message definition:

`log`	Optional string. Log message to add. If set, *layer* and *level* are mandatory.
`layer`	String. Layer name. Only mandatory if *log* is set.
`level`	String. Log level: *error*, *warn*, *info* or *debug*. Only mandatory if *log* is set.
`dir`	Optional string. Log direction: *UL*, *DL*, *FROM* or *TO*.
`ue_id`	Optional number. UE_ID.
`flush`	Optional boolean (default = false). If set, flushes fog file.
`rotate`	Optional boolean (default = false). If set, forces log file rotation.
`cut`	Optional boolean (default = false). If set, forces log file reset.

`log_reset`
: Resets logs buffer.

`quit`
: Terminates ltemonitor.

`help`
: Provides list of available messages in *messages* array of strings and events to register in *events* array of strings.

`stats`
: Report statistics for LTEMONITOR.
Every time this message is received by server, statistics are reset.
Warning, calling this message from multiple connections simultaneously will modify the statistics sampling time.
Response definition:

`cpu`
: Object. Each member name defines a type and its value cpu load in % of one core.

`instance_id`
: Number. Constant over process lifetime. Changes on process restart.

`counters`
: Object. List of counters, with following sub members:

`messages`
: Object. Each member name is the message name and its value is its occurence.
To get list of message, type *cevent help msg* in LTEMONITOR monitor.

`errors`
: Object. Each member name is the error name and its value is its occurence.
To get list of message, type *cevent help error* in LTEMONITOR monitor.

`register` Register client for messages generated by server. Message definition:

> `register` Optional string or array of string. List of messages to register to. Can be `components`
>
> `unregister`
> Optional string or array of string. List of messages to unregister. Can be `components`

8.6 LTE messages

`state_get`
Get components informations.

Response definition:

`components`
Object. Each property name represents component ID and property value its state:

`state`	String. Component state.
`id`	String. Component ID.
`name`	String. Component name.
`type`	String. Component type.
`info`	String. Additional information depending on state.

8.7 Remote events

Following events are sent by *PROG* if they have been registered on WebSocket.

`components`
Generated when any component state has changed.

`sender` Object. See [Components], page 24.

8.8 Examples

1. Config
   1. Client sends

```
{
    "message": "config_get",
    "message_id": "foo"
}
```

   2. Server replies

```
{
    "message_id": "foo",
    "message": "config_get",
    "name": "UE",
    "logs": {
        "phy": {
            "level": "error",
            "max_size": 0
        },
        ...
        "rrc": {
            "level": "debug",
            "max_size": 1
        }
    }
}
```

2. Error
    1. Client sends

    ```
    {
        "message": "bar",
        "message_id": "foo"
    }
    ```

    2. Server replies

    ```
    {
        "message_id": "foo",
        "message": "bar",
        "error": "Unknown message: bar"
    }
    ```

9 Log file format

9.1 EVENT layer

When an EVENT message is dumped, the format is:

```
time [layer] timestamp|hostname|level|component|section|title|message
```

Note that time of log may differ from event timestamp as event may come from a remote machine.

9.2 Other layers

```
time [layer] message
```

10 Change history

10.1 Version 2024-09-13

- `com_logs_lock` parameter is renamed to `com_log_lock`. `com_logs_lock` is still supported for backward compatibility
- `com_log_us` parameter is added

10.2 Version 2023-03-17

- `com_addr` parameter now uses [::] address instead of 0.0.0.0 in the delivered configuration file to allow IPv6 connection

10.3 Version 2022-12-16

- `utc` parameter is added to remote API response messages

10.4 Version 2022-09-16

- `priority` parameter is added to alarms

10.5 Version 2022-06-17

- `start_timestamp` and `end_timestamp` are added to `log_get` API

11 License

Abbreviations

Software component	Any Amarisoft component such as MME, IMS, eNB, MBMSGW or UE.