LTE License Server

Version: 2025-06-13

Table of Contents

1 Introduction

LTE License Server is a floating license server allowing to dispatch your Amarisoft LTE components licenses over multiple hardwares.

2 Features

- Share licenses over multiple hardwares.
- Allow use of VM for Amarisoft LTE/NR components.
- Command line monitor.
- Remote API using WebSocket.

3 Requirements

3.1 Hardware requirements

- Any PC, with a network connection, which is reachable from the other PCs running Amarisoft LTE/NR components (EPC, eNodeB, etc.) requiring a floating license from this server.

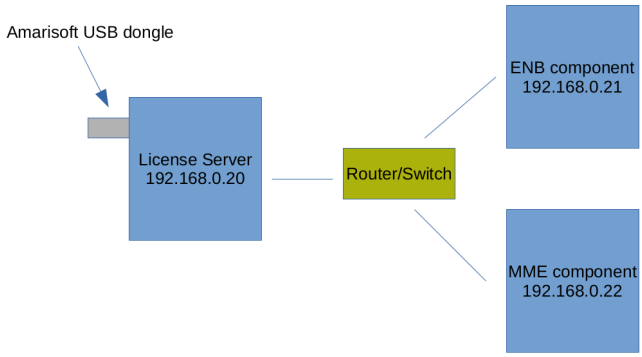
3.2 Software requirements

- 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.

4 License server installation

The picture below shows a possible license server configuration.



The license server can also run on one of the PC running Amarisoft LTE/NR components.

Amarisoft provides a USB dongle containing the license key file (`ltelicense.key`) required by LTELICENSE to run.
This is the only file that must be present under `/.amarisoft` folder of the USB dongle. Component license key files (ltemme.key, lteenb.key, etc..) must no be copied in this USB key.

The USB dongle has to be mounted on the PC which acts as license server. `Chapter 2.7 of installguide.pdf` explains how to mount a USB drive. Once the dongle is correctly mounted, the steps below can be followed:

- If not yet done, download the SW release tarball from your Extranet (amarisoft.YYYY-MM-DD.tar.gz)
- Untar it ("tar xzf amarisoft.YYYY-MM-DD.tar.gz") and copy the ltelicense-linux-YYYY-MM-DD.tar.gz from your release tarball to your license server PC
- Untar it ("tar xzf ltelicense-linux.YYYY-MM-DD.tar.gz"). This will create a directory called YYYY-MM-DD.
- Go to the directory YYYY-MM-DD and execute the following command with ROOT priviledges. The LTELICENSE can be run directly from the directory where it has been unpacked. No need for explicit installation.

```
./ltelicense_server config/license.cfg
```

You can also install it and make it run as a service using `install.sh` script and `OTS` provided within your release tarball.

At this point your LTELICENSE should be up and running.

4.1 Adding a floating license

When requesting to Amarisoft a new license key for any of your NR/LTE components, please ask for floating license.

Once you get the component license files (like lteenb.key, ltemme.key, ...), place them under `$HOME/.amarisoft/floating/` of the license server machine.

You can configure additional paths or change this path by editing the `licenses` array in the configuration file `config/license.cfg`, (see [licenses], page 10).

Configuration file example:

```
{
  log_options: "license.level=info,all.max_size=0,license.max_size=1",
  log_filename: "/tmp/license.log",

  bind_addr: "[::]",

  licenses: [
    "/root/.amarisoft/floating/"
  ]
}
```

When license file is installed, you can restart LTELICENSE or type `reload` in monitor.

4.2 Connecting a component to license server

Once the license server is up and running, you can now configure the Amarisoft components (i.e eNB, MME etc..) so they can connect to the server and retrieve the floating license.

- On the PC which runs the components, create a directory called `.amarisoft` under `${HOME}` where `${HOME}` is the home directory of the `root` user. (example /root/.amarisoft)
- Under this directory, create a license_server.cfg file and add the license server parameters :

  Example :

  ```
  {
      license_server: {
          server_addr: "192.168.1.20"
      }
  }
  ```

- server_addr value is the IP address of your license server, change it accordingly.
- Alternatively, it is also possible to define the `license_server` object inside each component's configuration file, please refer to lteenb.pdf and ltemme.pdf.

4.3 Adding multiple floating licenses

The license server can provide different license types to multiple setups.
To distinguish the licenses, the `tag` field can be used.
The `licenses` object in the configuration file `config/license.cfg` can look as the example below.

```
licenses:[
          {
            file: "/root/.amarisoft/floating_eNB600",
            tag: "eNB600"
          },
          {
            file: "/root/.amarisoft/floating_gNB200",
            tag: "gNB200"
          }
         ]
```

`tag` can be anything.

Each HW setup running a component (i.e eNB, MME etc..) has to be configured with a matching tag, as below:

First setup

```
license_server:{
            server_addr:192.168.0.20,
            tag: eNB600,
            }
```

Second setup

```
license_server:{
            server_addr:192.168.0.20,
            tag: gNB200,
            }
```

If the components run on the same HW the `license_server` object can be defined inside each configuration file (enb.cfg, mme.cfg, ...).

All license key files related to your license server (same code id) can be donwloaded by clicking on `"All"` link on your Extranet.

Once you have unzipped the tarball, place the corresponding key files under the paths indicated in the the `licenses` object.

4.4 Multiple LTELICENSE instances running on the same PC

A single license server can delivery floating licenses to many components, however if you have more than one USB dongles you can also run multiple instances of `ltelicense_server` on the same server.

In case of two USB dongles, mount also the second usb key and run a second license server, you should now have two instances of the follwing:

```
./ltelicense_server config/license.cfg
```

In the configuration of each license server (license.cfg) you need to define different port numbers in `bind_addr` and which license key file to use in `licenses` array, like the following:

First instance

```
{
   log_options: "license.level=info,all.max_size=0,license.max_size=1",
   log_filename: "/tmp/license_1.log",
   bind_addr: "[::]:9050",
   licenses: [
       {
         file: "/root/.amarisoft/ltemme-77-76-b7-9e-b8-da-bd-4d.key",
       }
   ]
}
```

Second instance

```
{
   log_options: "license.level=info,all.max_size=0,license.max_size=1",
   log_filename: "/tmp/license_2.log",
   bind_addr: "[::]:9051",
   licenses: [
       {
         file: "/root/.amarisoft/ltemme-61-77-c8-95-ae-a8-80-2d.key",
       }
   ]
}
```

Inside each component configuration file (like enb.cfg or mme.cfg) you need to set the license server port number you want to connect to, examples:

```
license_server:{
  server_addr:192.168.0.20:9050,
}
license_server:{
  server_addr:192.168.0.20:9051,
}
```

4.5 OpenSSL

LTELICENSE has been compiled against openssl version 1.1.1w.
If your system does not have compatible version installed you may have this error message at startup:

```
error while loading shared libraries: libssl.so.1.1: cannot open shared object file: No
```

To overcome this problem, you may:

- Copy libssl.so.1.1 and libcrypto.so.1.1 from `libs` subdirectory of your release tarball.
  If you have installed software with automatic install script, this should have been done automatically.
- Compile and install proper openssl version yourself

In case of persisting issue, raise a ticket from our support site at `https://support.amarisoft.com/` with the information provided by below commands executed in LTELICENSE directory:

```
uname -a
ls -l
ldd ./ltelicense
openssl version
```

5 Configuration reference

5.1 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 9, 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.

5.1.1 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 ],
}
```

5.2 Global properties

`bind_addr`

String. Set the IP address (and optional port) on which the license server will listen to component requests. The default port is 9050. It is normally the IP address of the network interface connected to other components.

`licenses`

Array. List of path or license files to use with this server.
Each member can be a string representing a path or a filename.
Path will be parsed recursively and all license inside will be used.
Member can also be an object with following parameters:

`file` String. Represent a path or filename.

`tag` Optional string. If set, server will allocate license to client with same tag field.

`com_addr` Optional string. Address of the WebSocket server remote API. See [Remote API], page 12.
If set, the WebSocket server for remote API will be enabled and bound to this address.
Default port is 9006.

Setting IP address to [::] will make remote API reachable through all network interfaces.

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

`com_ssl_certificate`
Optional string. If set, forces SSL for WebSockets. Defines CA certificate filename.

`com_ssl_key`
Optional string. Mandatory if *com_ssl_certificate* is set. Defines CA private key filename.

`com_ssl_peer_verify`
Optional boolean (default is false). If *true*, server will check client certificate.

`com_ssl_ca`
Optional string. Set CA certificate. In case of peer verification with self signed certificate, you should use the client certificate.

`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 14, 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).

`log_filename`
String. Set the log filename. If no leading `/`, it is relative to the configuration file path. See [Log file format], page 20.

`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: `license`

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

6 Remote API

You can access LTELICENSE 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`).

6.1 Messages

Messages exchanged between client and LTELICENSE 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.

6.2 Startup

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

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

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

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

```
{
    "message": "authenticate",
    "type": "LICENSE",
    "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": "LICENSE",
    "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.

6.3 Errors

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

6.4 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 LTELICENSE.
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 127.0.0.1:9006 '{"message": "config_get"}'
```

6.5 Common messages

`config_get`
: Retrieve current config.

  Response definition:

  `type` Always "LICENSE"

  `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 15).
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.

`license` Retrieves license file information.
Response definition:

`products` String. List of products, separated by commas.

`user` String. License username.

`validity` String. License end of validity date.

`id` Optional string. License ID.

`id_type` Optional string. License ID type. Can be `host_id` or `dongle_id`

`uid` Optional string. License unique ID.

`filename` Optional string. License filename.

`server` Optional string. License server URL.

`server_id`
: Optional string. License server ID.

`quit` Terminates ltelicense.

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

`stats` Report statistics for LTELICENSE.
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.

6.6 License messages

`reload` Force license reload.
If config file has been changed, modifications will be applied.

`list` Retrieve licenses states.
Response definition:

`licenses` Array of objects representing licenses with following properties:

- `uid` String. License unique ID
- `products` String. List of associated products separated by commas.
- `origin` String. License origin.
- `tag` Optional string. Associated tag
- `max` Integer. Maximum number of allowed connections.
- `version` String. License version limit.
- `connections` Array of object representing current connections:
  - `product` String. Associated product
  - `name` String. Name
  - `origin` String. Connection host and port

`message` Send message to all connected clients.
Message definition:

`text` String. Message to send.

7 Command line monitor reference

The following commands are available:

`help` Display the help. Use `help` *`command`* to have a more detailed help about a command.

`log [`*`log_options`*`]`
: Display the current log state. If *log_options* are given, change the log options. The syntax is the same as the `log_options` configuration property.

`msg` *`message`*
: Send a message to all connected component in their monitor window.

`msgto client_id message`
: Send a message to a specific client.

`client` List connected clients.

`close client_id [message]`
: Send a close message to a spcific clent, with an optionl text.

`list` List all license with their state.

`reload` Reload licenses from configured path. If config file has been changed, path will be updated has well.

`quit` Stop server.

8 Log file format

9 Change history

9.1 Version 2024-09-13

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

9.2 Version 2024-06-14

- OpenSSL library is upgraded to 1.1.1w
- added `clients`, `close` and `msgto` monitor commands

9.3 Version 2023-12-15

- added `com_ssl_ca` parameter for SSL verification

9.4 Version 2023-06-10

- `com_logs_lock` parameter added to disable logs configuration change via remote API

9.5 Version 2023-03-17

- `com_addr` and `bind_addr` parameters now use [::] address instead of 0.0.0.0 in the delivered configuration file to allow IPv6 connection
- `reload` command now handles configuration file modifications

9.6 Version 2022-12-16

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

9.7 Version 2022-06-17

- OpenSSL library is upgraded to 1.1.1n
- `start_timestamp` and `end_timestamp` are added to `log_get` API

9.8 Version 2021-12-17

- `license` monitor command is added

10 License

`ltelicense` is copyright (C) 2012-2025 Amarisoft. Its redistribution without authorization is prohibited.

`ltelicense` is available without any express or implied warranty. In no event will Amarisoft be held liable for any damages arising from the use of this software.

For more information on licensing, please refer to `license.pdf` file.