LTE and NR Network scanner

Version: 2024-12-23

Table of Contents

1 Introduction

LTESCAN is a tool using Amarisoft SDR50 boards to scan a band of the RF spectrum searching for either LTE or NR (SA) cells.
LTESCAN can be greatly accelerated by using several boards together.
Depending on conditions, adding a LNA RF amplifier in front of the SDR50 board(s) may give a better detection performance.
Launching “ltescan” will use the default config file “ltescan.cfg”. You can also specify a different config file by launching “ltescan myscan.cfg” Once launched, there are two ways to use it:

- user input on the console, with results on the screen
- websocket API for automatization

2 Requirements

2.1 Hardware requirements

- A fast PC:
  - For best performances, a quad core Intel Core i7 CPU (Haswell architecture or later) is recommended. Support of the AVX2 instruction set extension is required to run the software.
  - At least 2 GB of RAM.
  - At least 1 GB of hard disk space.
  - The video adapter does not matter.
- Radio front end

  one or more Amarisoft PCIe SDR50
- Appropriate antennas for the intended LTE frequencies or cables and attenuators to connect to a UE.
- Optional RF amplifier

2.2 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.

  Other distributions can be used provided the radio frontend drivers are available for them.

3 Installation

3.1 Linux setup

3.1.1 Packages

3.1.2 OpenSSL

LTESCAN 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 such file or directory
```

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 LTESCAN directory:

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

3.2 RRH setup

Please refer to sub section of your radio frontend to set it up.

3.2.1 Amarisoft PCIe SDR

Read the PCIe SDR documentation (`trx_sdr.pdf`).

3.3 LTESCAN installation

Decompress the LTESCAN archive to a convenient place. The executable `ltescan` can be launched from this directory.

4 Configuration reference

4.1 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 5, 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.

4.1.1 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 ],
}
```

4.2 Global properties

Example of ltescan.cfg:

```
/* LTE scan configuration */
{
  log_options: "all.level=debug,all.max_size=1,phy.signal=1",
  log_filename: "/tmp/scan.log",

  com_addr: "[::]:9009",

  rf_driver: {
      name: "sdr",
      args: ‘dev0=/dev/sdr0‘,
      /* For multiple boards */
      //args: ‘dev0=/dev/sdr0,dev1=/dev/sdr1,dev2=/dev/sdr2,dev3=/dev/sdr3‘,
  },

  rx_gain: 60,

  scan: {
    lock_timeout: 100,    /* ms: waiting for Cell Lock */
    pbch_timeout: 200,    /* ms waiting for PBCH */
```

```
    sib_timeout: 3000,    /* ms: waiting for SIB */

    snr_threshold: 3.0,   /* only show cells with SNR greater than 3dB */
    rssi_timeout: 1000,   /* RSSI integration time (ms) */
    rssi_width: 100,      /* RSSI block width for rssi results (KHz) */
  },
}
```

For UHD driver (B2x0 hardware), the rf_driver must be modified as follows:

```
  rf_driver: {
      name: "uhd",
      args: "num_recv_frames=64,num_send_frames=64",
      dl_sample_bits: 12,
      ul_sample_bits: 12,
  },
```

`log_filename`
: String. Set the log filename. If no leading `/`, it is relative to the configuration file path. See [Log file format], page 21.

`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. Rename current log with timestamp and open new one.
  - file.rotate=*size*. Rename current log every time it reaches *size* bytes open new one. Size is an integer and can be followed by K, M or G.
  - file.path=*path*. When log rotation is enabled, 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: `phy`

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

`rx_gain` Float. Receive gain in dB. The range is device dependent. For the PCIe SDR board, the range is between -11 and 77 dB (the exact limits depend on the RX frequency). For the USRP N2x0 device with the SBX daughterboard, the range is 0 to 31.5 dB. With an array of floats a different gain is specified for each channel. Ltescan uses this value as initial rx_gain fo each new frequenncy. The actual gain is automatically lowered as needed.

`scan` Object. Section for scanning parameters

`lock_timeout`
Number. Timeout in ms waiting for cell lock on a frequency. (default = 100ms)

`pcbh_timeout`
Number. Timeout in ms waiting for PBCH (LTE) or PDCCH (NR) when locked on a frequency. (default = 200ms)

`sib_timeout`
Number. Timeout in ms waiting for cell SIB (LTE) or MIB (NR) on a frequency when locked. (default = 3000 ms)

`snr_threshold`
Float. Minimum SNR in dB to accept SIB or MIB. (default = 3.0 dB)

`rssi_timeout`
Number. Time in ms integrating data when scanning RSSI (default = 100 ms)

`rssi_width`
Number. Frequency step in KHz when returning RSSI scan result. (granularity = 100KHz, default = 100 KHz)

`band` Number or String. Optional parameter to force automatic scan when ltescan is launched. Accepts same syntax as the 'scan' command on command line (see Chapter 6).

`sib_format_jer`
Optional boolean (default = false). If set, sib ASN.1 content within [Cell notification], page 17, message will be a JSON structure (JER) instead base64 encoding.

`com_addr` Optional string. Address of the WebSocket server remote API. See [Remote API], page 10.
If set, the WebSocket server for remote API will be enabled and bound to this address.
Default port is 9009.
Setting IP address to [::] will make remote API reachable through all network interfaces.

`com_name` Optional string. Sets server name. SCAN 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 12, 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).

`sim_events`
: Array of object. Each element gives an event configuration to execute for this UE. Event configuration is exactly the same as for [Remote API], page 10, messages except that message field must be event.

`sim_events_loop_count`
: If set, will define `loop_count` for each event of `sim_events`, See [loop_count], page 11.

`sim_events_loop_delay`
: If set, will define `loop_delay` for each event of `sim_events`, See [loop_delay], page 11.

`license_server`
: Configuration of the Amarisoft license server to use.
Object with following properties:

`server_addr`
String. IP address of the license server.

`name` Optional string. Text to be displayed inside server monitor or remote API.

`tag` Optional string. If set, server will only allow license with same tag.

Example:

```
license_server: {
```

```
    server_addr: "192.168.0.20",
    name: "My license"
}
```

5 Remote API

You can access LTESCAN 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.

5.1 Messages

Messages exchanged between client and LTESCAN 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.

5.2 Startup

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

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

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

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

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

5.3 Errors

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

5.4 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 LTESCAN.
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 127.0.0.1:9009 '{"message": "config_get"}'
```

5.5 Common messages

`config_get`
: Retrieve current config.

  Response definition:

  `type` Always "SCAN"

  `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 7,

    `max_size` See [log_options], page 7,

    `key` See [log_options], page 7,

`crypto` See [log_options], page 7,

`payload` See [log_options], page 7,

`count` Number. Number of bufferizer logs.

`rotate` Optional number. Max log file size 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.

`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 7.
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 7.

`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.

`quit` Terminates ltescan.

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

`stats` Report statistics for LTESCAN.
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.

`register` Register client to message generated by server. Message definition:

`register` String or array of string. List of message to register to.
Can be `cell`, `scan`

`unregister`
String or array of string. List of message to unregister.
Can be `cell`, `scan`

5.6 LTE messages

`scan` Launches a scan on selected bands.
The response is sent when scanning is over.
During scanning, notification intermediate message will be sent to notify progress.
Closing connection before end of scan does not stop scanning.
If a scanning is already in progress, it will be stopped. Message definition:

`band` Number, string, object or array. Defines the list of bands to scan.
If an array is defined, all elements must be a number, string or object.
A number represents a LTE band number to scan.
A string represents a single band with the command line syntax (see below).

`7` band LTE 7

`7(3000)` arfcn 30000 of band LTE 7

`n78(7838,7850)`
gscn 7838 to 7850 on band NR 78

An object will have following properties:

`band` Number. Band to scan

`type` String. Can be `nr` or `lte`.

Cell notification. Every time a cell is found, such notification is sent.

`notification`
String. Set as `cell`

`band` Number. Band

`type` String. Can be `nr` or `lte`.

`frequency`
Number. Cell center frequency in Hz.

`dl_earfcn`
Number. Downlink earfcn (LTE only).

`dl_arfcn` Number. Downlink arfcn (NR only).

`gscn` Number. SSB gscn (NR only).

`nsa` Optional boolean. Cell only supports NSA (no SIB1 scheduled, NR only).

`mib` String. Content of MIB in base64 encoding.

`bandwidth`
Number. Cell bandwidth in MHz.

`n_rb_dl` Number. Number of resource blocks.

`rssi` Number. RSSI in dBm.

`rsrq` Number. RSRQ in dB

`rsrp` Number. RSRP in dBm.

`snr` Number. SNR in dB.

`n_antenna_pbch`
Number. Number of antenna as reported in the PBCH (LTE only)

`lte_m` Boolean. Cell supports LTE-M (LTE only)

`n_ssb` Number. Number of SSB (antennas) from 'ssb-PositionsInBurst' in SIB1 (NR only)

`ssb_index`
Number. SSB index as reported in MIB (NR only)

`ssb_subcarrier_spacing`
Number. SSB subcarrier spacing in KHz (NR only)

`subcarrier_spacing`
Number. subcarrier spacing in KHz (NR only)

`uldl_config`
Number. uldl config if TDD mode (LTE only)

`special_subframe_config`
Number. special subframe config if TDD mode (LTE only)

`tp1_period`
Number. Period in ms of TDD Pattern_1 if TDD mode (NR only)

`tp1_dl_slots`
Number. Number of DL slots in Pattern_1 if TDD mode (NR only)

`tp1_ul_slots`
: Number. Number of UL slots in Pattern_1 if TDD mode (NR only)

`tp1_dl_symbs`
: Number. Number of DL symbols in special slot in Pattern_1 if TDD mode (NR only)

`tp1_ul_symbs`
: Number. Number of UL symbols in special slot in Pattern_1 if TDD mode (NR only)

`tp2_period`
: Number. Period in ms of TDD Pattern_2 if TDD mode (NR only)

`tp2_dl_slots`
: Number. Number of DL slots in Pattern_2 if TDD mode (NR only)

`tp2_ul_slots`
: Number. Number of UL slots in Pattern_2 if TDD mode (NR only)

`tp2_dl_symbs`
: Number. Number of DL symbols in special slot in Pattern_2 if TDD mode (NR only)

`tp2_ul_symbs`
: Number. Number of UL symbols in special slot in Pattern_2 if TDD mode (NR only)

`sib1` String. Content of SIB1 (if broadcasted). May be base64 encoding or JER depending on [sib_format_jer], page 8.

`sib2` String. Content of SIB2 (if broadcasted). May be base64 encoding or JER depending on [sib_format_jer], page 8.

`sib3` String. Content of SIB3 (if broadcasted). May be base64 encoding or JER depending on [sib_format_jer], page 8.

`sib4` String. Content of SIB4 (if broadcasted). May be base64 encoding or JER depending on [sib_format_jer], page 8.

`sib5` String. Content of SIB5 (if broadcasted). May be base64 encoding or JER depending on [sib_format_jer], page 8.

Band notification. Every time a new band is being scanned, such notification is sent.

`notification`
: String. Set as `cell`

`band` Number. Band

`type` String. Can be `nr` or `lte`.

`cells` Returns last scanning results. Response definition:

`cells` Array of object. Each item represents a cell as described in See [Cell notification], page 17.

`scanning` Boolean. If true, a scanning is in progress.

`scan_rssi`

Launches a RSSI scan on selected bands.
Results are returneded by frequency step (defined by rssi_width parameter). During

scanning, notification intermediate message will be sent to notify progress.
Closing connection before end of scan does not stop scanning.
If a scanning is already in progress, it will be stopped. Message definition:

`band` Number, object or array. Defines the list of bands to scan.
If an array is defined, all elements must be a number or an object.
A number represents a LTE band number to scan. An object will have following properties:

`band` Number. Band to scan

`type` String. Can be `nr` or `lte`.

RSSI notification. For every frequency step, a notification is sent.

`notification`
String. Set as `rssi`

`frequency`
Number. Frequency in Hz.

`arfcn` Number. Downlink arfcn (LTE only).

`rssi` Number. RSSI in dBm.

`rssis` Returns last RSSI scanning results. Response definition:

`rssis` Array of object. Each item represents a cell as described in See [RSSI notification], page 19.

`scanning` Boolean. If true, a scanning is in progress.

Exemple of scan command:

```
./ws.js -t 600 127.0.0.1:9009 '{"message": "scan","band":["7(3000)","n78"]}'
```

5.7 LTE events

Following events are sent by scanner to remote API client if they have been registered on WebSocket.

`cell` Cell has been scanned. Same definition as See [Cell notification], page 17.

`scan` Provides information on scanning progress

`notification`
String. Can be `band` when a new band is being scanned, `end` when scanning is over.

5.8 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"
    }
    ```

6 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.

`t [scan|cpu|spl] [`*`port`*`]`
: Activate various traces on the console. The display is stopped when typing return. The default trace is the current scan status. An optional RF port may be given for cpu and spl traces.

  Available traces:

  `scan` Display current scan information.

  `cpu` Display the CPU usage from the TRX (transceiver) API and the TX-RX latency statistics.

  `spl` Display various statistics about the sent and received complex samples (at the TRX API level). For the TX side, the RMS and maximum sample value are displayed. The number of saturation events (`abs(sample) > 1`) are displayed too. For the RX side the RMS and maximum sample value are displayed. The unit is dB FS (dB Full Scale). 0 dB FS is reached with a square signal of amplitude 1.

`scan` *`bands`*
: Start a cell scan on the specified band(s); LTE band: 1..256; NR band: n1..n256
  You can also add arfcn/gscn limits for each band. Examples:
  scan 7 20 will scan bands LTE 7 and 20
  scan 7(3000) scan arfcn 3000 on band LTE 7
  scan n78(7838,7850) scan gscn 7838 to 7850 on band NR 78

`scan_rssi` *`bands`*
: Start a RSSI scan on the specified band(s); LTE band: 1..256; NR band: n1..n256

`com` COM connection status.

7 Log file format

7.1 PHY layer

When a PHY message is dumped (debug level), the format is:

```
time layer dir ue_id cell rnti frame.subframe channel:short_content
   long_content
```

`time` Time using the selected format.

`layer` Layer (`[PHY]` here).

`dir` UL (uplink) or DL (downlink).

`ue_id` eNodeB UE identifier (hexadecimal, unique among all cells).

`cell` Low 8 bits of the cell identifier (hexadecimal).

`rnti` Associated RNTI (hexadecimal) or `-` if none.

`frame.subframe`
: Frame number (0-1023) and either subframe number (0-9) for LTE and NB-IoT cells or slot number for NR cells.

`channel` PHY channel name (e.g. PUSCH, PUCCH, PRACH, SRS, PSS, PBCH, PCFICH, PDSCH, PHICH, PDCCH, EPDCCH, ...).

`short_content`
: Single line content.

`long_content`
: Hexadecimal dump of the message if `phy.max_size > 0`.

8 Change history

8.1 Version 2024-12-13

- added `sib_format_jer` parameter

8.2 Version 2024-09-13

- added LTE bands 107 and 108 definition
- added `mib` parameter to cell notification
- 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

8.3 Version 2024-06-14

- OpenSSL library is upgraded to 1.1.1w
- added NR band 54 definition
- added `lte_m` and `nsa` parameters to cell notification
- added MIB info for WS API

8.4 Version 2024-03-15

- added LTE bands 106, 253 and 254 definition
- added NR bands 31, 72, 105, 109 and 254 definition

8.5 Version 2023-12-15

- added `loop_count` and `loop_delay` to remote API messages
- added `sim_events`, `sim_events_loop_count` and `sim_events_loop_delay`
- added `com_ssl_ca` parameter for SSL verification

8.6 Version 2023-06-10

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

8.7 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

8.8 Version 2022-12-16

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

8.9 Version 2022-06-17

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

8.10 Version 2022-03-18

- Added Introduction
- Added sample config file
- Use retransmission to improve all SIB decoding in low SNR conditions
- Indicate NSA cells
- Fixed cell detection when SSB spacing is different
- Fixed BWP parsing

9 License