EMSCOPE WEBSOCKET PROTOCOL DOCUMENTATION

This page documents the WebSocket protocol for EMScope **software ≥ 4.1**.

For software version 4.0, the protocol is identical except that measurement data was always sent as JSON (see section 7).

The Emscope software consists in a *WebSocket* server listening on *port 8010* of the machine and a *Web Server* listening on the standard HTTP *port 80* that serves the front end html and related files.

To configure, parametrize the measurements and manage the standards, a *JSON* style command set is used through a *WebSocket* connection.

1. First connection

Upon connection, the server does not send any data automatically. The client must first send a *session_UUID* message to activate itself. See section 8 for details on session locking.

The server responds to this with device info:

```
{
  "SN": "123456789",
  "MAC": "00:11:22:33:44:55",
  "SFP_SN": "SFP123456",
  "sw_version": "4.1",
  "measurement_uncertainty": "0.5 dB",
  "num_points": 8192
}
```

After that, the client must send *trace_type* to enable measurement data streaming. See section 6 for a full initial configuration example.

2. Default values

The default values for measurements are:

- Channel: LG
- Detector: PK
- Tracer: Clear/write
- Units: dbuv
- RWB: 9
- Subranges: 10
- Margin: 10
- Active: true
- Mode: circuit
- input_attenuator: auto
- reference_level: 115 (dbuV). Software 4.3 and earlier start at 100.
- average: 10

3. Commands and responses

The following parameters can be sent to configure the device:

Name	Parameter	Type	Default
Channel	measure_channel	“lg”, “ng”, “cm”, “dm” (RX2) / “l1”, “l2”, “l3”, “n”, “cm”, “dm1”, “dm2”, “dm3” (RX4)	“lg”
Detector type	detector_type	“pk”, “qp”, “av”	“pk”
Trace	trace_type	“clearwrite”, “maxhold”, “minhold”, “freeze”, “average”	“clearwrite”
RBW	rbw	“200”,“9”,“120” (CISPR), “1”, “10” (MIL), “200_9”, “1_10” (dual band)	“9”
Average (for trace average)	average	integer (10 - 20)	10
Mode	measure_mode	“circuit”, “modal”	“circuit”
Reference level	reference_level	integer	115
Input attenuator	input_attenuator	{0..78} (integer), “auto”	“auto”
Amplitude units	amp_units	“dbm”, “dbmv”, “dbuv”, “watts”, “volts”	“dbuv”
Serial number	SN	string	
Sweep time	sweep_time	{1..15} (float)	1
External loss attenuator	external_loss	External loss name (string)	none
Neutral (RX4)	neutral	true, false	false
Modal coefficients	modal_coefficients	object (see *modal_coefficients*)	

The channels “cm”, “dm”, “dm1”, “dm2” and “dm3” are measured with *measure_mode* “modal”; the other channels with “circuit”.

With *input_attenuator* “auto”, the attenuation follows *reference_level*: a higher reference level adds attenuation. Signals read the same, but the instrument's own noise floor reads higher (on a test instrument in band B: about 11 dBuV with 100, about 22 dBuV with 115). Send *reference_level* at the start of every session, so that measurements do not depend on the server default.

4. Changing RBW

After setting the RBW we shouldn't send any commands until the device it sends back the RBW value. This is so because a hot-change of firmware, that typically takes around 3.5 seconds, is required. To set a new RBW, send a **rbw** field with the string corresponding to the RBW:

```
{
  "rbw": "9", "threephase": false
}
```

RBW table:

Value	RBW	Frequency	Standard
200	200 Hz	9 kHz - 150 kHz	CISPR 16-1-1
9	9 kHz	150 kHz - 30 MHz	CISPR 16-1-1
120	120 kHz	30 MHz - 110 MHz	CISPR 16-1-1
1	1 kHz	10 kHz - 150 kHz	MIL-STD-461
10	10 kHz	150 kHz - 30 MHz	MIL-STD-461
200_9	200 Hz / 9 kHz	9 kHz - 30 MHz	Dual-band
1_10	1 kHz / 10 kHz	10 kHz - 30 MHz	Dual-band

threephase (boolean, default false) — set to true for three-phase (RX4) instruments and EMI_3P. Must be sent together with rbw in the same message. Sending rbw without threephase defaults to false.

5. Additional configuration fields

The following fields can be sent as individual JSON messages at any time after the RBW has been set.

display_range

Restricts the frequency window of the data sent by the server to a sub-range of the active band.

Send:

```
{ "display_range": [from_hz, to_hz] }
```

Both values are absolute frequencies in Hz and must lie within the band of the currently active RBW.

Example — receive only the 150 kHz – 30 MHz portion of the 9 kHz band:

```
{ "display_range": [150000, 30000000] }
```

To restore the full band, send the full frequency limits of the active band.

visible

Controls whether the trace is shown on the instrument's front panel display.

On RX4 instruments in three-phase mode the instrument measures only the inputs of the visible traces: a trace that is not visible receives data only when its channel is in the default pair (“l1”/“n” or “cm”/“dm1”). Send “visible”: true on every connection that measures.

Send:

```
{ "visible": true }
{ "visible": false }
```

trace_id

Names the trace of this connection in the rows of a “tabs” report (first column) and in *not_evaluated*, *overload_traces* and *source*. The web interface sends it for every tab; a client that does not send it gets null.

Send:

```
{ "trace_id": 1 }
```

neutral

RX4 instruments: tells the instrument whether the neutral conductor is connected. It selects the set of modal coefficients applied in three-phase modal measurements (“emi_3p_neutral” or “emi_3p_no_neutral”).

Send:

```
{ "neutral": true }
{ "neutral": false }
```

modal_coefficients

Defines each modal channel as a weighted sum of the instrument inputs, computed before the FFT: c1·In1 + c2·In2 + … It requires the **advanced_modal** license; without it the server ignores the message. The server does not reply to this message.

Configuration	Used when	Modal channels	Inputs c1, c2, c3, c4
emi	RX2	cm, dm	L, N
emi_3p_no_neutral	RX4, “neutral”: false	cm, dm1, dm2, dm3	L1, L2, L3
emi_3p_neutral	RX4, “neutral”: true	cm, dm1, dm2, dm3	L1, L2, L3, N

Each coefficient is a [real, imaginary] pair; both parts must be within -1 and +1 (+1 is applied as 0.99997). Only the configurations sent are changed, but every channel sent must carry all the coefficients of its configuration.

Example — common and differential mode of L1 and L2 against the neutral:

```
{
  "modal_coefficients": {
    "emi_3p_neutral": {
      "cm":  {"c1": [0.5, 0], "c2": [0, 0],   "c3": [0, 0], "c4": [0.5, 0]},
      "dm1": {"c1": [0.5, 0], "c2": [0, 0],   "c3": [0, 0], "c4": [-0.5, 0]},
      "dm2": {"c1": [0, 0],   "c2": [0.5, 0], "c3": [0, 0], "c4": [0.5, 0]},
      "dm3": {"c1": [0, 0],   "c2": [0.5, 0], "c3": [0, 0], "c4": [-0.5, 0]}
    }
  }
}
```

- The channel names are fixed; their content is free.
- The instrument measures two modal channels per sweep: cm with dm1, and dm2 with dm3. With more than two traces both pairs are measured in consecutive sweeps.
- With coefficients 1 and -1 a differential channel can reach twice the full scale of one input: it saturates and the trace shows OVERLOAD, as when an input overranges (before software 5.0: without an overload flag). The default channels use 0.5 and -0.5: add 6.02 dB to obtain the full difference.
- The coefficients apply to the RBWs up to 30 MHz; with RBW “120” the modal channels are fixed.
- A server restart returns them to the defaults: send them at the start of every session. Software 4.3 and earlier also lose them on an RBW change that loads another firmware (“120”, “osc” and back): there, send them again after the RBW.

freeze (trace_type)

In addition to the trace types listed in section 3, the value “freeze” is also accepted:

```
{ "trace_type": "freeze" }
```

The instrument continues sweeping internally but stops updating the displayed trace. Sending a different trace_type unfreezes it.

6. Initial configuration

A typical initial configuration parameters example would be:

```
{
  "detector_type": "pk",
  "measure_channel": "lg",
  "trace_type": "clearwrite",
  "amp_units": "dbmv",
  "rbw": "9",
  "reference_level": 70,
  "input_attenuator": "auto",
  "sweep_time": "1"
}
```

7 - Receiving measurements

To start receiving measurements, send at least **trace_type**. From that point the server streams data continuously after each sweep completes.

Since **SW 4.1**, measurement data uses a Binary WebSocket frame.

The **frequency axis** is sent as a separate JSON message and is cached — it is not repeated in every sweep frame.

7.1 - Frequency axis (JSON)

Sent by the server once when the RBW changes or the display range is updated. All subsequent sweep frames share this cached axis until the next such change.

```
{ "freq": [9000.0, 18000.0, 27000.0, ...], "rbw": "9" }
```

- `freq` — array of absolute frequencies in Hz, one per point of the spectrum.
- `rbw` — the active RBW string, echoed for reference.

The spectrum is interpolated: between two consecutive measurement bins there are three interpolated points, so every bin is a point of the axis and the step is a quarter of a bin (for example, about 1272 Hz in the 9 kHz band). With the whole band on screen this is about 23 475 points in the 9 kHz band and about 42 401 in the “200_9” dual band. Software 4.3 and earlier put one point between bins (about 11 739 and 21 203 points). Take the number of points from the length of `freq`; do not assume a fixed value.

7.2 - Binary amplitude frame (EMI mode, SW ≥ 4.1)

One binary WebSocket frame is sent per sweep. The layout is:

```
Offset  Size     Field
------  -------  ----------------------------------------
0       1 byte   flags
1       3 bytes  padding (ignored)
4       N×4 B    float32 amplitude values (little-endian)
```

flags byte:

- `flags & 0x01` → **overload**: the ADC that carries this trace saturated during this sweep. Reduce `reference_level` or increase `input_attenuator`.
- `flags & 0x02` → **repetition**: a quasi-peak repeat sweep was triggered (200 Hz and 9 kHz bands only).
- `flags & 0x08` → **dwell reduced**: the spectrogram fast mode measured this sweep with its own, shorter sweep time instead of the dwell time set. Quasi-Peak and Average values are uncalibrated while it is set.
- `flags & 0x10` → **overload held**: what this trace shows contains data measured with the ADC overloaded — the last sweep in clear write, any sweep of a max hold, min hold or average until it restarts, and it is kept by a frozen trace. Levels marked so are not valid for a compliance decision.

The number of float32 values (N) matches the length of the last received `freq` array. Each value corresponds to the amplitude at `freq[i]`, in the units configured via `amp_units`.

A sweep is not sent when the firmware could not measure it (a capture that failed, a processing error, or a three-phase input the firmware refused): the trace keeps its previous frame and no invalid data is merged into a hold. If the measurement firmware did not start after loading it, every client receives a `show_message` titled “`Measurement hardware`” and no frames arrive until the instrument is restarted.

7.4 - Input attenuator echo

When `input_attenuator` is set to "`auto`", the server sends the resolved hardware attenuation as a separate JSON message after each configuration change:

```
{ "input_attenuator": 30 }
```

This message is independent of the sweep data frame. It is not embedded in binary frames.

7. Locking the device to the current session

Sending *session_UUID* is required — the server will not activate the client or send any data until it receives this message. It also serves as a session lock: if another client already holds the lock with a different UUID, the server closes the connection with close code 1000 and close reason "4003" (check the reason, not the code). Use the same UUID for subsequent reconnections to the same session.

```
{
  "session_UUID": "12345qwerty"
}
```

The server responds immediately with device info (SN, MAC, SFP_SN). Once this exchange is complete, the client is active and can send configuration messages.

8. Standards (or Limits)

8.1 Getting the standards list

Send the field **get_standards** with a value of "true":

```
{
  "get_standards": true
}
```

It will return a **standards** object with an array of standards with the following format:

```
"Standard name": [ { "rbw": value, "data":
  [ From frequency (Mhz), To frequency (Mhz), From Quasi-peak (dBuV), To Quasi-peak (dBuV),
Average from Quasi-peak (dBuV), Average to Quasi-peak (dBuV) ],
  ...
  }
]
```

For example:

```
{
  "standards": [
    {
      "Standard One": { "rbw": "9", "data": [
        [1, 2, 3, 4, 5, 6],
        [7, 8, 9, 10, 11, 12],
        ...
      ]
      }
    },
```

```
    {
      "Standard Two": { "rbw": "200", "data": [
        [21, 22, 23, 24, 25, 26],
        [27, 28, 29, 210, 211, 212],
        ...
      ]
      }
    },
    ...
  ]
}
```

8.2 Creating a standard

To create a standard, send a **name** field with the name of the new standard, a **modify** field set to “false” and a **values** array in the same format we described before when receiving the standards list.

```
{
  "name":"New standard",
  "modify":false,
  "standard_rbw": "10",
  "values":[
    ["1","2","3","4","5","6"],
    ["11","12","13","14","15","16"],
    ...
  ]
}
```

8.3 Editing a standard

Send the same values as creating it, but with the field *modify* set to *true* and an additional field, *original_name*, just in case we wanted to rename the original *name*, i.e:

```
{
  "original_name": "New standard",
  "name":"Modified standard",
  "modify":true,
  "standard_rbw": "1",
  "values":[
    ["21","22","23","24","25","26"],
    ["11","12","13","14","15","16"],
    ...
  ]
}
```

8.4 Deleting a standard

Simply send a **delete_standard** with the name of the standard, i.e:

```
{
  "delete_standard": "Standard One"
}
```

8.5 Resetting standards to factory defaults

Simply send a **reset_standards** with a value of “true”:

```
{
  "reset_standards": true
}
```

9. Sessions (or Presets)

9.1 Getting the sessions list

Send the field **get_sessions** with a value of “true”:

```
{
  "get_sessions": true
}
```

It will return a **sessions** object with an array of sessions with the following format:

```
[
"Session name": [BLOB DATA],
  ...
]
```

For example:

```
{
  "sessions": [
    {
                                                                "Session One":
"{\"PROGRAM_VERSION\":1.25,\"id_counter\":2,\"current_center_freq_units\":\"mhz\",\"current_st
art_freq_units\":\"mhz\",\"current_stop_freq_units\":\"mhz\",\"current_span_units\":\"mhz\",\"
current_center_frequency\":15000000,\"current_start_frequency\":0,\"current_stop_frequency\":3
0000000,\"current_span\":30000000,\"measure_mode\":\"circuit\",\"current_display_scale\":0,\"a
Detectors\":[{\"type\":\"normal\",\"measure_channel\":\"lg\",\"trace_type\":\"clearwrite\",\"d
etector_type\":\"pk\",\"amp_units\":\"dbuv\",\"color\":\"red\",\"hidden\":false,\"standard\":n
ull,\"id\":0,\"sweep_time\":1},{\"type\":\"normal\",\"measure_channel\":\"lg\",\"trace_type\":
\"clearwrite\",\"detector_type\":\"pk\",\"amp_units\":\"dbuv\",\"color\":\"red\",\"hidden\":fa
lse,\"standard\":null,\"id\":1}],\"reference_level\":100,\"reference_level_volts\":0.1,\"ampli
tude_step\":10,\"aCurrentFreq\":[150000,30000000],\"aCurrentSpan\":[0,100],\"aMarkers\":[[],[]
],\"aMMarkers\":[],\"aStandards\":{\"CISPR 22 CLASS
A\":[[\"0.15\",\"0.5\",\"79\",\"79\",\"66\",\"66\"],[\"0.5\",\"30\",\"73\",\"73\",\"60\",\"60\
"]],\"CISPR 22 CLASS
B\":[[\"0.15\",\"0.5\",\"66\",\"56\",\"56\",\"46\"],[\"0.5\",\"5\",\"56\",\"56\",\"46\",\"46\"
],[\"5\",\"30\",\"60\",\"60\",\"50\",\"50\"]]},\"aStandardsQuasiPeak\":{},\"aStandardsAverage\
":{},\"active_tab\":1,\"ZERO_SPAN_MODE\":false,\"amp_units\":\"dbuv\",\"rbw\":\"9\",\"input_at
tenuator\":\"auto\",\"sweep_time\":1,\"current_color_theme\":\"dark\"}",
        ...
      ]
    },
    {
      "Session Two": "......",
        ...
      ]
    },
```

```
    ...
  ]
}
```

9.2 Creating a session

To create a session, send a **session_name** field with the name of the new session, a **modify** field set to “false” and a **session_values** array in the same format we described before when receiving the sessions list.

```
{
  "session_name":"New session",
  "modify":false,
  "session_values": ".....",
    ...
  ]
}
```

9.3 Editing a session

Send the same values as creating it, but with the field *modify* set to *true* and an additional field, *original_session*, just in case we wanted to rename the original *name*, i.e:

```
{
  "original_session": "New session",
  "session_name":"Modified session",
  "modify":true,
  "values": "......",
    ...
  ]
}
```

9.4 Deleting a session

Simply send a **delete_session** with the name of the session, i.e:

```
{
  "delete_session":"Session One"
}
```

9.5 Deleting all sessions

Simply send a **reset_sessions** with a value of “true”:

```
{
  "reset_sessions": true
}
```

10. External loss attenuators

10.1 Getting the external loss attenuators list

Send the field **get_external_losses** with a value of “true”:

```
{
  "get_external_losses": true
}
```

It will return an **external_losses** object: for each external loss attenuator, its name and its rows as a JSON string. Each row is `[ Frequency (MHz), Line (dB), Neutral (dB) ]`; values may come as strings:

```
{
  "external_losses": {
    "External loss attenuator 1": "[[\"0.15\",\"10\",\"10\"],[\"30\",\"20\",\"20\"]]",
    "External loss attenuator 2": "[[\"0\",\"5\",\"5\"],[\"30\",\"5\",\"5\"]]"
  }
}
```

10.2 Creating an external loss attenuator

To create an external loss attenuator, send an **external_loss_name** field with the name of the new external loss attenuator, a **modify** field set to “false” and a **values** array of rows `[ Frequency (MHz), Line (dB), Neutral (dB) ]` (an array, not a string).

The correction is interpolated in logarithmic frequency between the rows:

- the rows are sorted by frequency when the table is stored;
- a row at 0 MHz means “from DC” and is used at 1 Hz, where the chart draws it;
- a negative or repeated frequency, or a value that is not a number, makes the server refuse the table: it is not stored and the client receives a `show_message` with the reason.

The same applies to the three-phase tables and to the transfer impedance tables.

```
{
  "external_loss_name":"New External loss attenuator",
  "modify":false,
  "values":[
    ["1","2","3"],
    ["10","20","30"],
    ...
  ]
}
```

10.3 Editing an external loss attenuator

Send the same values as creating it, but with the field *modify* set to *true* and an additional field, *original_external_loss_name*, just in case we wanted to rename the original *external_loss_name*, i.e:

```
{
  "original_external_loss_name": "New External loss attenuator",
  "external_loss_name":"Modified External loss attenuator",
  "modify":true,
  "values":[
    ["11","22","33"],
    ["21","22","23"],
    ...
```

```
  ]
}
```

10.4 Deleting an external loss attenuator

Simply send a **delete_external_loss** with the name of the external loss attenuator, i.e:

```
{
  "delete_external_loss": "External loss attenuator 1"
}
```

11. Report lists

A report compares the measurement with the limits of a standard and lists the emissions found. Send, in one message:

```
{
  "standard": "CISPR 32 CLASS B band B",
  "subranges": 10,
  "margin": 10,
  "freq_start": 150000,
  "freq_stop": 30000000,
  "report_type": "tabs",
  "report_image": false,
  "scale_div": 10,
  "markers": {},
  "only_markers": false
}
```

- *standard*: name of the standard (limits) to compare with.
- *subranges*: number of sub-ranges (log-spaced) the scan is split into; in each one the emission closest to its limit, or the one over it the most, is reported.
- *margin*: emissions more than *margin* dB below the limit are not listed.
- *freq_start*, *freq_stop*: range to evaluate, in Hz. **Required**: without them the server reads 0-0 Hz and sends no report.
- *report_type*: "`tabs`" evaluates the traces on screen (one per connection, each with its own detector and trace type); "`full`" takes a new acquisition with Peak, Quasi-Peak and Average on every channel.
- *only_markers*: evaluate only the marker frequencies (*markers*: per trace id, a list of frequencies in Hz).

The answer has a *report* array with one row per emission. For "`tabs`" each row is:

```
[
 Trace id (string, or null when the connection did not send trace_id),
 Marker (integer),
 Frequency in Hz (float),
 Peak level, Peak limit, Distance to Peak limit [dB],
 Quasi-Peak level, Quasi-Peak limit, Distance to Quasi-Peak limit [dB],
 Average level, Average limit, Distance to Average limit [dB],
 Channel (string),
 Compliance ("PASS", "FAIL", or "-" when the row was not compared with a limit)
]
```

Levels and limits are in the units set with *amp_units*; a field that does not apply is an empty string. For "`full`" the row is the same without the trace id.

With the report come:

- *verdict*: "`FAIL`" if any emission is over its limit; "`PASS`" only if at least one trace was compared with a limit over the whole scan, nothing is over it and nothing was measured with overload; otherwise "`NOT_EVALUATED`" — nothing was

compared (no standard, or no limit for the measured detectors), only the marker frequencies were checked, the data was measured with the ADC overloaded, or the measurement failed (no answer in time, an error, or a capture or processing that failed in the firmware: the reason is given in *not_evaluated* or *error*). **"NOT_EVALUATED" is never a pass.**

- *detectors*: the detectors really compared with a limit, e.g. `["QP", "AV"]`.
- *not_evaluated*: what was measured but not compared, as `{"trace_id", "detector", "reason"}` entries.
- *overload_traces*: trace ids (`null` for the whole full report) measured with the ADC overloaded.
- *source* (`"tabs"` report): the traces evaluated, `{"type": "traces", "traces": [{"trace_id", "detector", "trace", "frozen"}]}`.
- *error*: present when the measurement for a full report failed; the report is then empty and `"NOT_EVALUATED"`.
- *frase*: **true** when nothing is over the limit and 6 or fewer emissions are listed within *margin* dB of it.

For example:

```
{
  "report": [
    [ "1", 1, 300064.08, "", "", "", 99.38, 60.24, 39.14, "", "", "", "L", "FAIL" ],
    ...
  ],
  "verdict": "FAIL",
  "detectors": ["QP"],
  "not_evaluated": [],
  "overload_traces": [],
   "source": {"type": "traces", "traces": [{"trace_id": "1", "detector": "QP", "trace": "clearwrite", "frozen": false}]},
  "frase": false
}
```

12. Keepalive

The server sends periodic keepalive messages to detect disconnected clients:

```
{ "ping": true }
```

The client MUST reply immediately with:

```
{ "pong": true }
```

If the client does not respond within the timeout, the server will close the connection. Clients that use the official Python API (EmscopeConnection) have this handled automatically.

13. Miscellaneous

13.1 Getting the activated licenses

Send the field **get_licenses** with a value of "true":

```
{
  "get_licenses": true
}
```

As a response you will get a **licenses** object with an array with the name of each activated licenses:

```
{"licenses": [ "emi", "osc" ]}
```

13.2 Getting the current PCB and FPGA temperature

Send the field **get_temps** with a value of “true”:

```
{
  "get_temps": true
}
```

As a response you will get a **temperatures** object with an array with the celsius degrees. The first value is for the PCB, the second one for the FPGA:

```
{"temperatures": [ 45.12345, 50.12345 ]}
```

From:
https://emzer.com/wiki/ -

Permanent link:
https://emzer.com/wiki/doku.php?id=public:emscope:websocket_api

Last update: **2026/11/04 17:53**