LEA Open API - Websocket Protocol

Rev 2. 10-08-2025

Introduction

Websockets are used as the transport for the LEA WebUI. Websockets are convenient since they combine the reliability of TCP with the notion of message boundaries. The LEA Connect Series products utilize JSON-RPC protocol and can be controlled and monitored from any device that is capable of sending and receiving JSON-RPC messages. The LEA Connect Series products offer an Open API via websockets, so any parameter that can be controlled and/or monitored via the WebUI can be controlled and monitored by any other device that communicates via websockets.

The LEA Connect Amplifiers use port 1234 for websocket communication

Any LEA Connect Firmware version accepts Websocket communication, however new features will be added in the future and will be included in future firmware versions.

Patterned after JSON-RPC 2.0 (https://www.jsonrpc.org/specification)

- the 'jsonrpc' member is replaced with an 'leaApi' member
- an additional 'url' member is required in requests and non-error responses
- the 'id' member, when present in requests, must be an integer number in the range [0, INT32_MAX]
- batch mode messages are not supported

LEA amplifiers typically act in the server role, while UIs and other devices act in the client role. It is also possible for devices to interact peer-to-peer, either over a single web/tcp socket or a pair of web/tcp sockets.

LEA Open API - Websocket Protocol

Rev 2. 10-08-2025

An example LEA Websocket API message looks like this. In this case, a request to set an amp's channel 1's mute status from mute off (False) to mute on (True)

Websocket Command Sent:


```
{
 "leaApi": "1.0",
 "url": "amp/channels/1/output",
 "method": "set",
 "params": {"mute":true},
 "id": 1
}
```


Websocket Response confirming Mute Command successful


```
{
 "leaApi": "1.0",
 "url": "amp/channels/1/output",
 "result": "OK",
 "id": 1
}
```


Example Websocket Message Element

{ ⟵ Opening bracket

"leaApi": "1.0", ⟵ (Should always start with this)

"url": "amp/channels/1/output", ⟵ (where the control and monitor command you want to interact with resides)

"method": "set", ⟵ (What do you want to do? Set, Get, Subscribe, Unsubscribe, and Info)

"params": "mute":true}, ⟵ (The parameter and what you want to do with it)

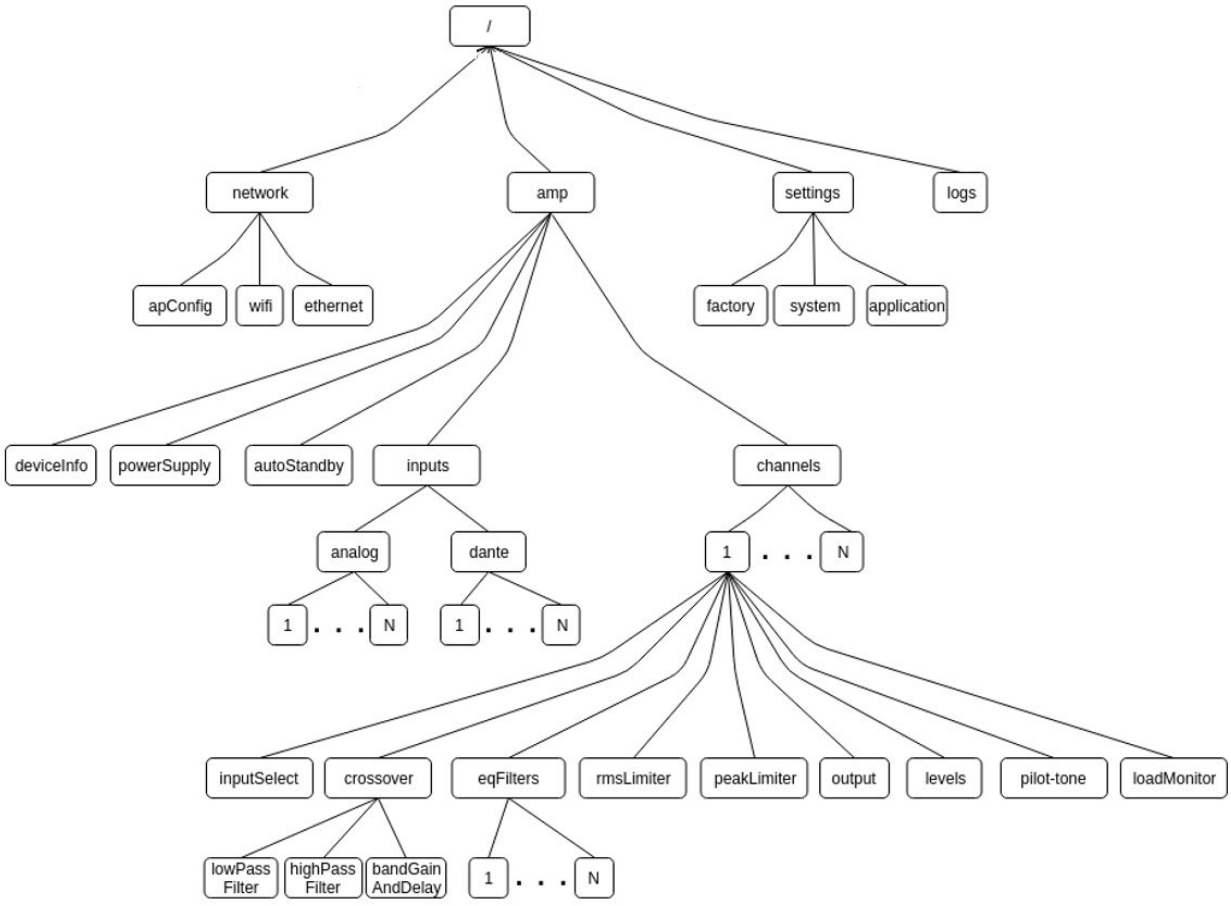
"id": 1 ⟵ (Should always end with this)

} ⟵ Closing bracket

Amp Object Hierarchy

Resources are divided into chunks called objects. At the top of the tree diagram above is the root object, specified by 'url'="/". Objects farthest way from the root on a given branch are called leaf objects. An example of a leaf object in the diagram is the object with 'url'="/amp/channel1/output" at bottom-right. All of the resources in a server are contained in the leaf objects. The root and other non-leaf objects are just containers for other objects. In terms of the API, this means that methods operate on leaves.

Many leaf objects support another level of hierarchy in the form of set of 'elements' that be accessed via the message parameters. For such objects, there is a one-to-one correspondence between parameters and elements. Essentially all of the audio processing objects are constructed this way.



LEA Open API - Websocket Protocol

Rev 2. 10-08-2025

Websocket Info Requests:
All objects support the 'info' method, which can be used to obtain information about the object, including the methods it supports, the elements it contains and the various element attributes (type, min/max, units, etc.). A list of all parameters in the LEA Connect Series Amplifiers can be found here: https://leaprofessional.com/wp-content/uploads/2023/02/LEA-Connect-All-Websocket-Objects-Oct-2025.pdf

To obtain **all** parameters of the Connect Series Amplifiers, simply send this websocket request:

```
{
 "leaApi": "1.0",
 "url": "/",
 "method": "info",
 "params": {},
 "id": 1
}
```

To obtain all the parameters for just the “amp” portion, simply send this websocket request:

```
{
 "leaApi": "1.0",
 "url": "/amp",
 "method": "info",
 "params": {},
 "id": 1
}
```

To obtain all the parameters for just the “input” portion of the “amp,” simply send this websocket request:

```
{
 "leaApi": "1.0",
 "url": "/amp/inputs",
 "method": "info",
 "params": {},
 "id": 1
}
```

Websocket SET Requests:

After you have received the parameter's info, it will tell you what each parameter supports from a Get, Set, Subscribe or Unsubscribe standpoint. Here's an example from the /amp/inputs info request:


```
{
 "leaApi": "1.0",
 "url": "/amp/inputs",
 "result": {
  "dante": {
   "1": {
    "methods": [
     "get",
     "set",
     "subscribe",
     "unsubscribe"
    ],
}
```


In this case you can Get Channel 1 Dante information, you can Set Channel 1 Dante info, you can Subscribe to Channel 1 Dante so if this changes, you will be notified, and if you've subscribed to this parameter, you can unsubscribe.

Here's an example to Get, Set, Subscribe and Unsubscribe to Channel 1 Output Level: Channel 1 output level is located here /amp/channels/1/output. If we do an info request for this url, we get the following information back from the amplifier for the output fader:


```
  "fader": {
   "type": "Float",
"control": true,
   "min": -80.0,
   "max": 0.0,
   "default": 0.0,
   "multipleOf": 0.10000000149011612,
   "units": "dB"
```

Websocket SET Requests (continued):

This is saying the fader has a max value of 0dB, a min value of -80dB with increments of 0.1dB. If I want to set the channel 1 output fader to -3.0dB, I send this SET command:


```
{
 "leaApi": "1.0",
 "url": "/amp/channels/1/output",
 "method": "set",
 "params": {"fader":-3.0},
 "id": 1
}
```


The amplifier responds with the following websocket message to let you know channel 1 output fader has been set to -3.0dB:


```
{
 "leaApi": "1.0",
 "url": "/amp/channels/1/output",
 "result": "OK",
 "id": 1
}
```


Another option for increasing or decreasing a channel output fader by a specific increment is through using a BUMP method. If I wanted to increase the channel 1 output fader back to 0dB from the previous SET command of -3.0, I could send this BUMP command:


```
{
"leaApi": "1.0",
"url": "/amp/channels/1/output",
"method": "bump",
"params": {"fader":3},
"id": 1
}
```

The amplifier responds with the following websocket message to let you know channel 1 output fader has been set back to 0dB:


```
{
 "leaApi": "1.0",
 "url": "/amp/channels/1/output",
 "result": "OK",
 "id": 1
}
```


This section covers the topic of sending a SET command to select a specific input source on a channel, including the option to select the input mixer and how to send commands to control the mixer levels.

To select the input mixer as the primary input source on channel 1, I would send this SET command:


```
{
 "leaApi": "1.0",
 "url": "/amp/channels/1/inputSelector/",
 "method": "set",
 "params": {"primary":"Mixer"},
 "id": 1
}
```


To control the individual input source levels within the mixer, the desired MixFader must be identified in the “params” line of the command. If I wanted to adjust the MixFader level for Analog1 to -40dB I would send this SET command:


```
{
 "leaApi": "1.0",
 "url": "/amp/channels/1/inputSelector",
 "method": "set",
 "params": {"analog1MixFader":-40},
 "id": 1
}
```

To access control of the level of the other input sources, the MixFader name will need to be altered to include the input source name that you want to adjust. For example: If I wanted to adjust the level of Dante 1 to -40dB within the mixer, I would alter the “params” line to say: {“dante1MixFader”:-40}

There is a MixFader for each of the analog inputs available on the amplifier being used, this number will vary depending on the number of channels within that model. There’s also a MixFader for each of the 8 Dante inputs. (Half-rack models have 4 Dante inputs) Commands can be sent to adjust the level of each of these by altering the MixFader name accordingly.

Below are examples of the SET command used to select Analog 1 or Dante 1 as the primary input source for channel 1:

```
{
 "leaApi": "1.0",
 "url": "/amp/channels/1/inputSelector/",
 "method": "set",
 "params": {"primary":"Analog 1"},
 "id": 1
}

{
 "leaApi": "1.0",
 "url": "/amp/channels/1/inputSelector/",
 "method": "set",
 "params": {"primary":"Dante 1"},
 "id": 1
}
```

Websocket GET, SUBSCRIBE, and UNSUBSCRIBE Requests:

To get the value for the channel 1 output to confirm the change was made, type the GET command for channel 1 output:


```
{
 "leaApi": "1.0",
 "url": "/amp/channels/1/output",
 "method": "get",
 "params": {},
 "id": 1
}
```


Once you send this GET Command, you receive the following information back from the amplifier and you can confirm the output fader:


```
{
 "leaApi": "1.0",
 "url": "/amp/channels/1/output",
 "result": {
  "name": "OutputName",
  "ZoneName": "ZoneName",
  "enable": true,
  "status": "Ok",
  "mute": true,
  "fader": -3.0,
  "zoneFader": 0.0,
  "backpanelPotAttn": 0.0,
  "super8": false,
  "hiZLoZ": "HiZ-100V",
  "hiZHpfEnable": true,
  "hiZHpfFrequency": 50.0,
  "clipLimiterEnable": false,
  "fault": false,
  "thermal": false,
  "limiting": false,
  "clip": false,
```

```
   "signalDetect": false,
   "ready": true,
   "autoStandbyActive": false
  },
  "id": 1
}
```


The channel 1 output fader object above supports methods 'get', 'set', 'subscribe' and 'unsubscribe'. The 'get' and 'set' messages need little explanation, except to note that the use of 'get' is strongly discouraged in other than development and test scenarios. In place of 'get', production clients should use 'subscribe', which is described later. Here are some representative 'get' and 'set' requests, along with server responses:

The subscription messages serve two purposes:

- The values of some elements representing measured quantities (aka 'sensors') are updated asynchronously when new measurement values are obtained (e.g. an audio level meter). A client uses subscriptions to request server notifications when such updates occur.
- In a multi-client environment, changes made by one client need to be seen by other clients. The subscription mechanism is used to propogate such changes.

Server notifications contain the 'notify' method. Here is an example of a client subscribing to a particular object:

'subscribe' request


```
  {
   "leaApi": "1.0",
   "url": "/amp/channels/1/rmsLimiter",
   "method": "subscribe",
   "params": {},
   "id": 13
  }
  {
   "leaApi": "1.0",
```

```
  "url": "/amp/channels/1/rmsLimiter",
  "result": {
   "enable": false,
   "threshold": 151.0,
   "attackTime": 0.019999999552965164,
   "releaseTime": 0.0010000000474974513,
   "gainReduction": -80.0,
   "totalGainReduction": -80.0
  },
  "id": 13
 }
```


subsequent changes to the object result in server notifications

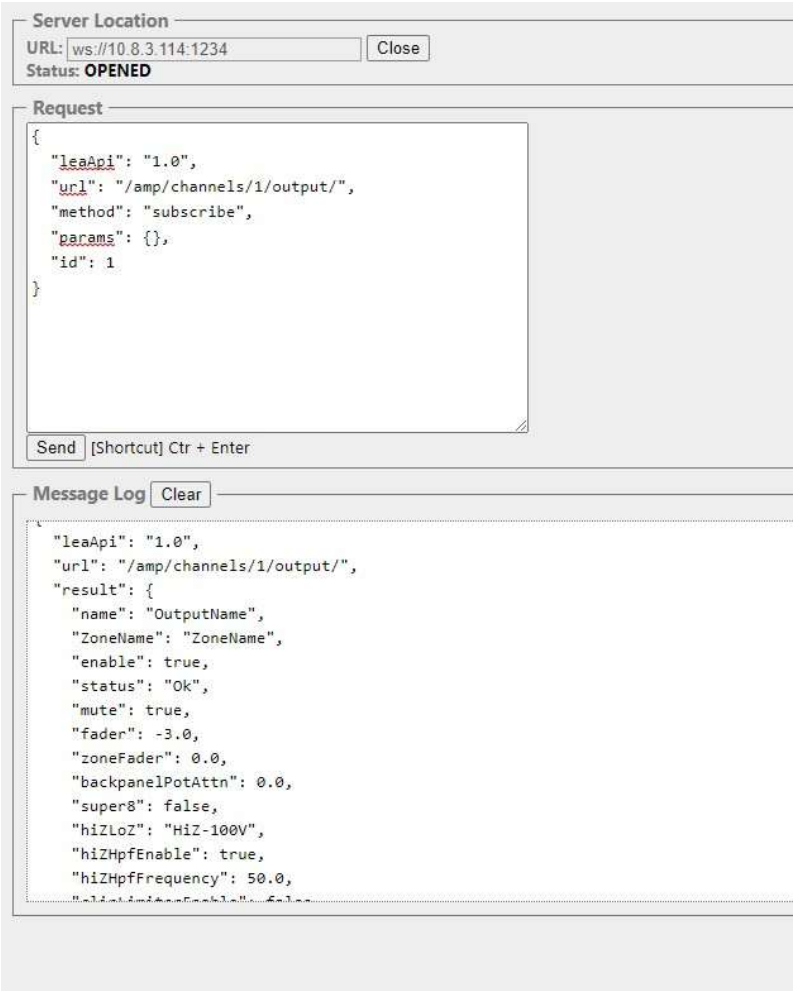

```
 {
  "leaApi": "1.0",
  "url": "/amp/channels/1/rmsLimiter",
  "method": "set",
  "params": {
   "threshold": 153.0
  }
 }
 {
  "leaApi": "1.0",
  "url": "/",
  "method": "notify",
  "params": {
   "amp": {
    "channels": {
     "1": {
      "rmsLimiter": {
       "threshold": 153.0
      }
     }
    }
   }
  }
```

```
  }
'unsubscribe' request
  {
    "leaApi": "1.0",
    "url": "/amp/channels/1/rmsLimiter",
    "method": "unsubscribe",
    "params": {},
    "id": 14
  }
  {
    "leaApi": "1.0",
    "url": "/amp/channels/1/rmsLimiter",
    "result": "OK",
    "id": 14
  }
```

Testing Websocket Commands from your Computer.

A good tool to use to test websockets from your computer is the WebSocket Test Client for Google Chrome Browsers:

https://chromewebstore.google.com/detail/websocket-test-client/fgponpodhbmadfljofbimhhlengambbn



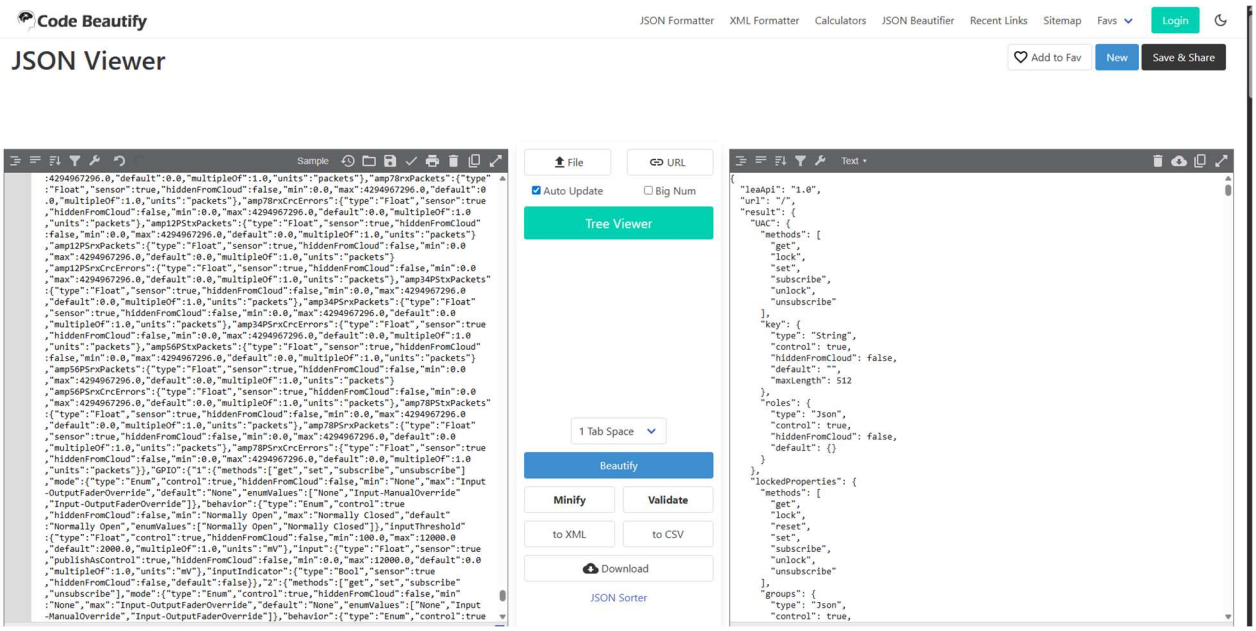
LEA Open API - Websocket Protocol

Rev 2. 10-08-2025

Arranging the all parameters response received from the amplifier.

A good tool for arranging the full API response received from the amp is the Code Beautify JSON Viewer. This online tool will convert the single line response received in the WebSocket Test Client to the same format as shown in this document.

https://codebeautify.org/jsonviewer



For any questions, please contract techsupport@leaprofessional.com