

API v2.0

Version 2.0 of the signalAPI, a websocket and HTTP API for control and feed back for all of the signal equipment

- [Websocket Connections](#)
- [HTTP Connections](#)
- [Message Structure](#)
- [JavaScript Wrapper Library](#)
- [Modules](#)

Websocket Connections

The Sygnal API is primarily websocket based. You can establish a connection to a sygnal device via the configured IP and port of the webGUI.

By the nature of being a websocket API there is no direct request-response model. However, in most cases when sending a request you should expect a single response. This response may be just to you, or to all clients if your request caused a change other clients are subscribed to receiving events about.

For example, if you send a request to start a stream, this will then start the stream and then send an update to ALL connected API clients with the new state of the stream. In most cases when creating, updating or deleting an item this change will be sent to all clients that are subscribed, it will in most cases send an updated list of all items to give clients an opportunity to recover if there has been de-sync. For example, if you create a new stream, a list of all streams including the new one will be sent to all clients.

The general structures of the Sygnal applications are split into 'modules'. For example the teleprompter is its own module and all communication about it are flagged as part of the 'prompt' module. When opening a new API connection, you must 'register' as part of this you specify the modules you would like to send/receive messages from. This means you could write a plugin/extension for playback and not receive the unnecessary prompt messages. You could also use this as a way of separating out your communication or even multithreading workflows. There is no limit to subscribers to each module or to the origin of the subscribers. Meaning you could open 3 different prompt API connections, all 3 would receive all the prompt messages, but each one could be responsible for sending a different type of message.

There is a defined structure and methodology to the websocket and HTTP message structure outlined in the message structure section.

HTTP Connections

The HTTP API follows the same format as the websocket API, in fact it used the same handler internally!

All requests should be sent as POST requests and the message should be sent JSON encoded as the body of the request.

Message Structure

The websocket and HTTP messages/requests have a defined structure that must be followed when both sending and receiving a message from a unit/server.

A message has two components, a header with some meta data and the payload, with the contents of the message. All keys in this structure are camelcase.

Header [object]

The header of the message contains 7 fields as follows:

Field	Type	Structure	Purpose	Example
fromID	string	First letter of 'type' _ 'timestamp of device initialisation' _ version	the ID of the original sender of the message	S_1707151018347_1.0.9
timestamp	number	unixtimestamp	The millisecond unix timestamp of when the message was sent	1707160733725
version	string	version.major.minor	Semantic version of the API version in use	1.0.9
type	string	Capitalised string	The type of the sender, to help identify what the purpose of the device is	Server/Browser
active	boolean	true/false	To determine if the original sending device is still online on the network. The message could be a relayed then cached message	true/false
messageID	number	unixtimestamp	The ID of the original message, this is typically the unixtimestamp at the time of sending, but could be any number. For detecting message duplication	1707160733725

Payload [object]

The structure of the payload is less set as there are options based on usage.

Module [string]

There will always be a 'module' field in all messages except the register message, as the register message is unique.

The module field is used to determine what module the message is from/for. This field is a string

Current valid options are: prompt, playback, system, config, network

Command [string]

Like module, there will always be a command field, event register has the command field!

The contains the actual 'command' you are sending to the module, e.g. set, update, delete e.t.c.

Data [object/any]

In most cases there will be a 'data' property which contains the actual data. In many case this is a series of key value pairs, however, in some cases that can be a large complex object and in others simply just a long string.

Register

To register you send a message the typical header structure and payload with the command "register" and a payload block which should be an object containing modules names you wish to send and receive messages from with the value true for receive and false to ignore. As follows:

```
{
  "header": {
    "fromID": "B_1707160733672_0.1.9",
    "timestamp": 1707160733738,
    "version": "0.1.9",
    "type": "Browser",
    "system": "SignalEnc-XXXX",
    "active": true,
    "messageID": 1707160733738
  },
  "payload": {
```

```
"module": "core",
"command": "register",
"data": {
  "flows":true,
  "network":true,
  "system":false,
  "prompt":true
}
}
```

Ping/Pong

There is a background ping/pong every second run by the server to all it's attached clients. This should be responded to or your client will be marked as inactive and you will stop recieving messages. If this occurs, sending a new message should re-activate your connection. If you are inactive for too long your connection will be closed by the server.

A ping message will contain a single command with the value "ping", it should be responded to with a single command "pong" as seen bellow:

```
{
  "header": {
    "fromID": "S_1707151018347_0.1.9",
    "timestamp": 1707160734618,
    "version": "0.1.9",
    "type": "Server",
    "active": true,
    "messageID": 1707160734618,
  },
  "payload": {
    "command": "ping"
  }
}
```

Examples:

Setting the system name

```
{
  "header":{
    "fromID":"S_1707151018347_1.0.9",
    "timestamp":1707160733725,
    "version":"1.0.9",
    "type":"Server",
    "active":true,
    "messageID":1707160733725,
  },
  "payload":{
    "module":"system",
    "command":"config",
    "data": {
      "key":"systemName",
      "value":"SygnalEnc-XXXX"
    }
  }
}
```

Recieveing a flow

```
{
  "header": {
    "fromID": "S_1707151018347_0.1.9",
    "timestamp": 1707160734618,
    "version": "0.1.9",
    "type": "Server",
    "active": true,
    "messageID": 1707160734618,
  },
  "payload": {
    "module": "flows",
    "command": "status",
    "data": {
      "name": "Hardware Encoder #1",
      "ID": "800fcb9b-7627-42be-b447-30c137a828ef",
      "type": "Hardware-Encoder",
      "flowType": "ENCODE",
      "profile": {
        "codec": "h265",

```

```
"profile": "66",
"bps": "",
"bpsMin": "",
"bpsMax": "",
"gop": "",
"rcMode": "0",
"rotation": "0",
"qpInit": "26",
"qpMax": "0",
"qpMin": "0",
"qpStep": "-1"
},
"enabled": false,
"active": false,
"caps": {},
"debug": false
}
}
}
```

Ping

```
{
  "header": {
    "fromID": "S_1707151018347_0.1.9",
    "timestamp": 1707160734618,
    "version": "0.1.9",
    "type": "Server",
    "active": true,
    "messageID": 1707160734618,
  },
  "payload": {
    "command": "ping"
  }
}
```

Receiving the current network interface info


```
{
  "header": {
    "fromID": "S_1707151018347_0.1.9",
    "timestamp": 1707160733723,
    "version": "0.1.9",
    "type": "Server",
    "active": true,
    "messageID": 1707160733723,
  },
  "payload": {
    "module": "network",
    "command": "ifaces",
    "data": [
      {
        "ifindex": 1,
        "ifname": "lo",
        "flags": [
          "LOOPBACK",
          "UP",
          "LOWER_UP"
        ],
        "mtu": 65536,
        "qdisc": "noqueue",
        "operstate": "UNKNOWN",
        "group": "default",
        "txqlen": 1000,
        "link_type": "loopback",
        "address": "00:00:00:00:00:00",
        "broadcast": "00:00:00:00:00:00",
        "addr_info": [
          {
            "family": "inet",
            "local": "127.0.0.1",
            "prefixlen": 8,
            "scope": "host",
            "label": "lo",
            "valid_life_time": 4294967295,
            "preferred_life_time": 4294967295
          },
          {
```

```
    "family": "inet6",
    "local": "::1",
    "prefixlen": 128,
    "scope": "host",
    "valid_life_time": 4294967295,
    "preferred_life_time": 4294967295
  }
]
},
{
  "ifindex": 2,
  "ifname": "enP2p33s0",
  "flags": [
    "BROADCAST",
    "MULTICAST",
    "UP",
    "LOWER_UP"
  ],
  "mtu": 1500,
  "qdisc": "mq",
  "operstate": "UP",
  "group": "default",
  "txqlen": 1000,
  "link_type": "ether",
  "address": "7a:b2:4e:29:01:79",
  "broadcast": "ff:ff:ff:ff:ff:ff",
  "permaddr": "3e:9f:56:b0:fe:c0",
  "addr_info": [
    {
      "family": "inet",
      "local": "192.168.1.29",
      "prefixlen": 24,
      "broadcast": "192.168.1.255",
      "scope": "global",
      "dynamic": true,
      "noprefixroute": true,
      "label": "enP2p33s0",
      "valid_life_time": 2775,
      "preferred_life_time": 2775
    }
  ],
}
```

```
{
  "family": "inet6",
  "local": "2a0d:3344:1d6:9210:3f:39e3:590c:a04b",
  "prefixlen": 64,
  "scope": "global",
  "temporary": true,
  "dynamic": true,
  "valid_life_time": 1194,
  "preferred_life_time": 594
},
{
  "family": "inet6",
  "local": "2a0d:3344:1d6:9210:bbab:1a3:272a:8c49",
  "prefixlen": 64,
  "scope": "global",
  "dynamic": true,
  "mngtmpaddr": true,
  "noprefixroute": true,
  "valid_life_time": 1194,
  "preferred_life_time": 594
},
{
  "family": "inet6",
  "local": "fdb4:be22:9e07:10::f5b",
  "prefixlen": 128,
  "scope": "global",
  "dadfailed": true,
  "tentative": true,
  "noprefixroute": true,
  "valid_life_time": 4294967295,
  "preferred_life_time": 4294967295
},
{
  "family": "inet6",
  "local": "fdb4:be22:9e07:10:cb85:5d3d:4f:f4ac",
  "prefixlen": 64,
  "scope": "global",
  "temporary": true,
  "dynamic": true,
  "valid_life_time": 600378,
```

```
"preferred_life_time": 81483
},
{
  "family": "inet6",
  "local": "fdb4:be22:9e07:10:d098:9a1e:6396:72be",
  "prefixlen": 64,
  "scope": "global",
  "mngtmpaddr": true,
  "noprefixroute": true,
  "valid_life_time": 4294967295,
  "preferred_life_time": 4294967295
},
{
  "family": "inet6",
  "local": "fe80::4408:7058:8fe9:54f9",
  "prefixlen": 64,
  "scope": "link",
  "noprefixroute": true,
  "valid_life_time": 4294967295,
  "preferred_life_time": 4294967295
}
],
"gateway": "192.168.1.1"
},
{
  "ifindex": 3,
  "ifname": "enP4p65s0",
  "flags": [
    "BROADCAST",
    "MULTICAST",
    "UP",
    "LOWER_UP"
  ],
  "mtu": 1500,
  "qdisc": "mq",
  "operstate": "UP",
  "group": "default",
  "txqlen": 1000,
  "link_type": "ether",
  "address": "46:62:0d:1d:e2:5b",
```

```
"broadcast": "ff:ff:ff:ff:ff:ff",
"permaddr": "aa:04:f5:fd:4e:35",
"addr_info": [
  {
    "family": "inet",
    "local": "192.168.88.223",
    "prefixlen": 24,
    "broadcast": "192.168.88.255",
    "scope": "global",
    "dynamic": true,
    "noprfixroute": true,
    "label": "enP4p65s0",
    "valid_life_time": 419,
    "preferred_life_time": 419
  },
  {
    "family": "inet6",
    "local": "fe80::51d7:4511:97ea:b968",
    "prefixlen": 64,
    "scope": "link",
    "noprfixroute": true,
    "valid_life_time": 4294967295,
    "preferred_life_time": 4294967295
  }
],
"gateway": "192.168.88.1"
},
{
  "ifindex": 4,
  "ifname": "wIP3p49s0",
  "flags": [
    "BROADCAST",
    "MULTICAST",
    "UP",
    "LOWER_UP"
  ],
  "mtu": 1500,
  "qdisc": "noqueue",
  "operstate": "UP",
  "group": "default",
```

```
"txqlen": 1000,
"link_type": "ether",
"address": "a0:02:a5:bd:d4:36",
"broadcast": "ff:ff:ff:ff:ff:ff",
"addr_info": [
  {
    "family": "inet",
    "local": "192.168.1.39",
    "prefixlen": 24,
    "broadcast": "192.168.1.255",
    "scope": "global",
    "dynamic": true,
    "noprfixroute": true,
    "label": "wIP3p49s0",
    "valid_life_time": 3373,
    "preferred_life_time": 3373
  },
  {
    "family": "inet6",
    "local": "2a0d:3344:1d6:9210::f5b",
    "prefixlen": 128,
    "scope": "global",
    "dynamic": true,
    "noprfixroute": true,
    "valid_life_time": 994,
    "preferred_life_time": 394
  },
  {
    "family": "inet6",
    "local": "2a0d:3344:1d6:9210:996e:cd23:3952:2114",
    "prefixlen": 64,
    "scope": "global",
    "temporary": true,
    "dynamic": true,
    "valid_life_time": 1194,
    "preferred_life_time": 594
  },
  {
    "family": "inet6",
    "local": "2a0d:3344:1d6:9210:3f13:80eb:c42b:56d3",
```

```
"prefixlen": 64,
"scope": "global",
"dynamic": true,
"mngtmpaddr": true,
"noprefixroute": true,
"valid_life_time": 1194,
"preferred_life_time": 594
},
{
  "family": "inet6",
  "local": "fdb4:be22:9e07:10::f5b",
  "prefixlen": 128,
  "scope": "global",
  "noprefixroute": true,
  "valid_life_time": 4294967295,
  "preferred_life_time": 4294967295
},
{
  "family": "inet6",
  "local": "fdb4:be22:9e07:10:9cd6:5d24:63b6:896a",
  "prefixlen": 64,
  "scope": "global",
  "temporary": true,
  "dynamic": true,
  "valid_life_time": 600319,
  "preferred_life_time": 81343
},
{
  "family": "inet6",
  "local": "fdb4:be22:9e07:10:90c1:4b8f:4d19:68e0",
  "prefixlen": 64,
  "scope": "global",
  "mngtmpaddr": true,
  "noprefixroute": true,
  "valid_life_time": 4294967295,
  "preferred_life_time": 4294967295
},
{
  "family": "inet6",
  "local": "fe80::a750:aedd:b0d8:d2c2",
```

```
    "prefixlen": 64,  
    "scope": "link",  
    "noprefixroute": true,  
    "valid_life_time": 4294967295,  
    "preferred_life_time": 4294967295  
  }  
],  
  "gateway": "192.168.1.1"  
}  
]  
}  
}
```


JavaScript Wrapper Library

There is a JavaScript ES6 wrapper library that can be imported and contains a class that can be used to simplify the communication.

For access this reach out on developer@signal.tv

Class

Constructor

```
serverURL, type, clientVersion, currentSystem, ssl = false, logger = console
```

Methods

setSystem(string)

close()

send(object)

makeHeader(): object

connect(string)

Events

open

close

error: error

message: header, payload, event

disconnect

ping

Examples

An example of the library being imported and used when it is in the /modules/ subdirectory is as follows:

```
import _ServerSocket from './modules/serverSocket.js';

const serverURL = '127.0.0.1';
const APIVersion = '1.0.0';
const systemName = 'Demo';
const isSSL = false;
const type = 'Browser';

const Server = new _ServerSocket(serverURL, type, APIVersion, systemName, isSSL);

Server.addEventListener('message', event => {
  const [header, payload] = event.detail;
  socketDoMessage(header, payload);
});

Server.addEventListener('open', event => {
  Server.send({
    'command': 'register',
    'subscribe': ['flows', 'network', 'system', 'prompt']
  });
  document.getElementById('disconnected').classList.add('hidden');
});

Server.addEventListener('close', ()=>{
  const _logs = document.getElementById('logs');
  _logs.innerHTML = '<div class="logDisconnect">Disconnected</div>' + _logs.innerHTML;
  document.getElementById('disconnected').classList.remove('hidden')
});

function socketDoMessage(header, payload) {
  switch (payload.module) {
    case 'flows':
      doFlows(payload);
      break;
    case 'network':
      doNetwork(payload);
      break;
  }
}
```

```
    case 'prompt':
        doPropmt(payload);
        break;
    case 'playback':
        doPlayback(payload);
        break;
    case 'system':
        doSystem(payload);
        break;
    default:
        break;
}
}
```

```
function doFlows(payload) {
    //doStuff
}
```

```
function doNetwork(payload) {
    //doStuff
}
```

```
function doPropmt(payload) {
    //doStuff
}
```

```
function doPlayback(payload) {
    //doStuff
}
```

```
function doSystem(payload) {
    //doStuff
}
```

Modules

Core [core]

core - requests

The core module is only really used to register your connection

Command	Data	Description
register	object: {module:bool}	Used to register your connection and recieve messages from the specified modules
ping	-	
pong	-	

System Information [sysInfo]

sysInfo - requests

The system infromation module can be used to retrieve information about the device, it's IO and current status, including temperatures

Command	Data	Description
io	-	Retrieves the list of IO and details about that IO
info	-	Retrieves the board info from the device
rate	int {time - ms}	Sets the polling rate for the system info watcher
magewellInfo	-	Returns info about the specified SDI source

api

system

gpio

joystick

ina219

licence

gstreamer

flows