WING

PERSONAL MIXING CONSOLE

WING Remote Protocols:
OSC remote control, MIDI SYSEX, Binary Interfaces,
and **wapi,** an API for WING

[V 2.1 - Wing FW 2.1 and above]

Table of Contents

Introduction

Introduction

About this document

My name is Patrick-Gilles Maillot and I am authorized by Behringer to publish and maintain this “WING remote protocols” document; I am not a MusicTribe employee.

Starting with release 2.0 of the WING firmware, OSC and native remote protocols form a single (this) document under two separate sections, and share the same series of appendix chapters.

Most users will probably find it easier to remote access their WING with **OSC** commands while more advance programing control is possible using **native** commands and the Wing API (**wapi**) library.

Dedicated chapters provide additional details on MIDI, Custom Controls, Effects, Plugins, Shows, Snap and Snippets and much more.

I want to thank the Behringer dev team for their continuous support in writing this document.

General features of the WING console

In 2019, Behringer has been designing a whole new digital mixing desk they would later call “personal mixing console”. The WING was unveiled to the public in November 2019 and first shipments took place in December. As to why calling it a “personal Mixing Console”, here is a perfectly valid answer from one of the fathers of the console: “A fundamental idea of WING was providing a high level of customization options to the engineer, allowing to adapt the console surface to his personal preferences and needs”.
The WING console was awaited by several X32 and M32 users as it carried the promise of new features, long expected since the first release the X32 and M32 family of digital mixing desks. It seems the WING receives a warm welcome from the community.

The Behringer WING provides 48-channel, 28-bus mixing with 24 motorized faders and a large 10” capacitive-touch LED screen. The desk is designed for live performance, live and studio recording, touring sound, A/V, club installs, and more. Three separate fader sections and a custom controls section can be easily and intuitively tailored to personal requirements.

The 48-channel inputs [in/aux] and 28-channel mixes [bus/matrix/main] can all be in mono/stereo or mid-side mode, with specific source mutes and metering, and provide dynamics, EQ and FX processing. They too can be given a color, icon, name and up to 8 console or user defined tags for grouping and filtering purposes.
WING input channels provide low-cut & high-cut filters, tilt-EQs, all-pass or Sound Maxer, in addition to a 6-band parametric EQ. All buses, matrices, and mains feature 8-band parametric EQ. All channels and buses can also load high-end simulations modeled from hardware devices such as Pultec EQ, SSL Bus Compressor and Gate/Expander, SPL Transient Designer, Neve EQ, Compressor and Gate, Focusrite ISA and D3, DBX160, LA-2A, 1176, Elysia mPressor, Empirical Labs Distressor, and more. The built in FX rack supports 8 true stereo processors including TC VSS3 algorithms, Lexicon, Quantec, and EMT emulations. Other processing includes modulation, equalization, dynamics, nonlinear effects and four guitar amplifiers with cabinet simulations. A maximum of 16 stereo inserts can be used for applying internal FX or outboard processing to input channels or buses.

The channel editing section provides instant channel status overview and flow of operation. It allows working on the selected channel processing, even when the main display is used for something completely unrelated. Touch-sensitive rotary controls allow you to display the most relevant information, all at your fingertips.
The central Custom Controls section offers user-assignable controls including 4 rotary encoders and 20 buttons with 2 LCDs that can be set as functions readily available. A big rotary wheel offers fine-adjustments for up to 8 user parameters or can be used for DAW remote control via USB MIDI. The control configuration also includes predefined functionality for USB and SD-card recorder transport, show control and mute groups.
WING includes 8 original MIDAS PRO microphone preamps and 8 XLR outputs with professional quality specifications. 8 TRS line auxiliary ins and outs help bring in signals from media players or computers. A brand new StageCONNECT[1] interface allows connecting breakout boxes and delivers up to 32 channels of low-latency input or output over a single standard XLR microphone cable.

WING can accommodate 376 inputs[2] and 374 outputs thanks to 3 AES50 SuperMAC audio networking ports, which connect to digital stageboxes. In addition, 144 input and 144 output streams can be shared with other mixing consoles. There are 48 channels of USB audio and 64 channels of Audio over IP (AoIP module optional), plus AES/EBU stereo I/O. The WING expansion card slot features the LIVE SD recording card with 64x64 channels of audio or can accommodate option cards for various standards such as ADAT, MADI, DANTE, and WSG.
All digital processing takes place on 40-bit floating point Digital Signal Processors, at 48 or 44.1 kHz, with a 1ms round-trip latency.
WING provides MIDI In/Out and 2x2 GPIO (General Purpose Input Output) that can be used as console event triggers and external show controls.
Automixing is also implemented, with 2 groups of gain sharing on any 16 input channels. The management of the respective input channel gains depends on the levels received, reducing the sum gain in the group to maintain intelligibility and low noise during meetings, ideal when several speakers are collaborating to corporate events, panels, broadcast applications or house of worship.

[1] https://www.klarkteknik.com/series.html?category=R-KLARKTEKNIK-STAGECONNECTSERIES
[2] Considering 4 USB stick IN/OUT capability, but not User Signal entries which overlap with actual sources

Sources vs. Inputs

Unlike many digital or analogue desks, WING makes a clear separation between Sources and Input channels; Historically, consoles focus on input numbers assigned to Channels and Auxes. WING is offering a different perspective by focusing on the Source as the reason for any mixing. Sources can be in mono, stereo, or mid-side mode, own headamp parameters like gain and phantom power, with specific source mute and metering. They can be given a color, icon, name and up to 8 console or user defined tags for grouping and filtering purposes. All of this describes the actual Source first, before being patched to Input channels which focus on processing or mixing.

This patching process (also called "Routing") is described in a specific "user-guide" like Appendix later in this document to help new users grasp the basic operations involved in assigning a physical source (local or remote) to a channel for mixing, as well as assigning WING processed audio data to a physical output (local or remote).

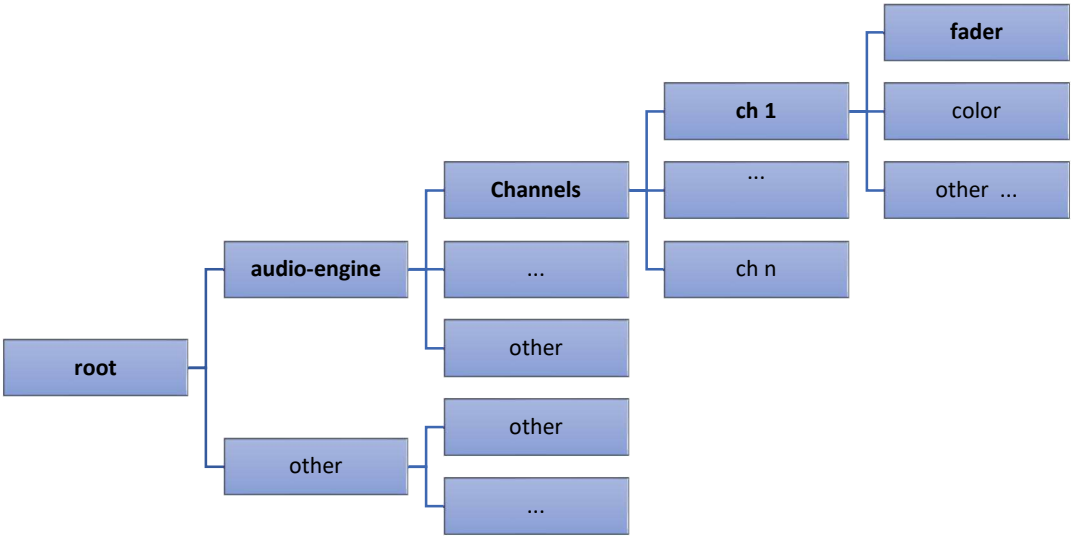
Sources can be labelled using the WING Co-Pilot app or other means such as OSC protocol described later in this document or the wapi function calls also presenter in the upcoming chapters[3], and no matter if the signal is patched to a channel, to SD recording or to any other output, it can always be referred to as its assigned Source label.

Notes
The internal real-time clock (RTC) is powered by a super-capacitor. If the WING is powered off for more than about two weeks it will most likely lose its clock data.

[3] Refer also to https://github.com/pmaillot/wapi

WING Internal Data

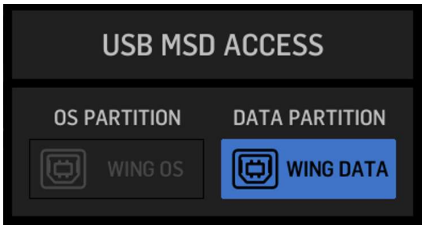
Like all digital or programmable devices, WING relies on an internal set of parameters that are stored/saved in non-volatile memory. This enables you to find the console in the same state you left it when powering it OFF. WING data set is very large, and in line with the many features the console offers. Each button, each attribute, color setting, effect, parameter, etc. can be found as an internal variable, member of a hierarchical tree structure.
The WING tree is more than 25000 elements! To organize this large set of internal variables, WING uses a hierarchical tree of data, stating with a root and dispatching parameters into logical groups (sub-trees or branches) until the last element (leaves) that represent the actual parameter.
For example, the `fader` associated to `channel 1` is part of the `channels` sub-tree, and is one of the many attributes of channel 1. The channel sub-tree is part of the `audio-engine`, itself at the `root` level.
A quick representation would be as shown below:



Computers use specific data structures to represent trees. WING uses one of them, based on `JSON`[4] notation. It is important to know/understand the list of sub-trees (nodes), and leaves (parameters) WING contains as this is how you can access to data. More detail on the WING data set is provided in appendix.

WING File System

At the difference of the X32, WING can be directly connected to a computer via USB; There are two ways WING can be visible to your computer, depending on the setting of the `SETUP→GENERAL` screen (shown below, with WING connected as an active data partition):



[4] JavaScript Object Notation: an efficient way to represent structured objects. Also used as a data-interchange format.

When actively connected to your PC either as an OS partition or a Data partition, the status at the top of the WING screen will show a red OS or DATA tag.

OS partition

WING can be seen as an `OS PARTITION`, a directory where you can deposit the FW release you will use to boot from at next power up or reboot. Use with caution!

Data Partition

A USB connected WING presents itself as an external disk drive. Therefore, the standard cautions apply when connecting and more important, disconnecting from the computer; ***Ensure you unmount the WING file system to avoid losing data.***

If the choice for `USB MSD ACCESS` in `SETUP→GENERAL` is set to `DATA PARTITION`, the WING file system will show as a standard external USB drive. There may be some folders already there, such as 'global' or 'shows', with subfolders, such as 'global/ch_presets', 'global/fx_presets', 'global/routing_presets', 'global/snapshots'.

Remote communications with WING

WING communicates via ports `2223 [UDP]`, and `2222 [UDP, TCP];`
Initiating a communication with WING starts with sending the 5 bytes `[UDP]` datagram '`WING?`' to the `IP` of your WING, port `2222`.

WING will reply to the requesting `IP` and `port` with the following datagram:

'WING,' [c_ip] ',' [c_name] ',' [c_model] ',' [c_serial] ',' [firmware]
where
[c_ip] e.g., '192.168.1.62'
[c_name] ascii characters
[c_model] 'ngc-full' (standard Wing console)
[c_serial] serial number (ascii)
[firmware] version string (ascii)

For its native communication format, WING proposes 14 'communication channels' to enable separation between the different 'engines' or main blocks of the console. The following communication channels are currently in use:
`Control-engine` (a `TCP` communication channel) is using channel #1
`Audio-engine` (main `TCP` communication channel) is using channel #2
`Meters` (`UDP` communications) are using channel #3

OSC uses a single communication port: `2223`

Keeping connections alive

Open connections will time out after `10 seconds` of inactivity (on the receiving side). One way to keep a connection active is to request at regular intervals of less than 10 seconds some data from the console. There are many data that can be collected, as shown later in this document.

Number of simultaneously connected applications

WING can simultaneously communicate with up to 16 connected 'clients'; The console will reject further connection requests, if the maximum number of simultaneous connections `(16)` is reached.
What we call 'clients' above refer to actual `TCP` ports that communicate with the console. Some applications may use several ports and this will reduce the actual number of applications that can simultaneously connect and communicate with WING.
`UDP` communications such as used for OSC do not have this limitation, being "connection-less". WING's OSC remote protocol enables only one (1) subscription to data (for receiving event messages) at any given time.

Subscriptions must be kept alive; they automatically die after 10 seconds.

Accessing WING Internal Data and Functions from remote programs

As mention in the introduction, WING hosts an OSC compliant remote protocol server that offers access to the full set of features of the desk. This is described in the "WING OSC protocol data interface" chapter below.

WING also offers a native, binary protocol with the capability to access (read or write) parameters of its internal structures and take full advantage of the entire set of features of the digital desk, including remote control. The protocol is fully described in the "**WING native/binary data interface**" chapter below. To help users access the native protocol, a WING API written in C [**wapi**] has been developed and is available at https://github.com/pmaillot/wapi to write programs that directly call **wapi** functions.

WING OSC protocol data interface

OSC Remote Protocol

WING includes an OSC Remote Protocol server. This enables easy access to remote features for many professional, sound applications and extensions offered by third parties.
OSC remote control enables reading and modifying (when possible) all parameters included in the `ae_data` and `ce_data JSON` structures, all part of the main parameter tree.

WING OSC server implementation complies with the OSC standard[5] and proposes several ways to access data, parameters, and features. As all OSC compliant servers, the WING OSC server runs in the console and will reply to UDP on a specific port: **2223**.
When using standard `UDP` communication, clients will be replied onto their calling port. If needed, a specific feature enables WING to reply to a `UDP port` specified by the connected client, as explained later in this document.

OSC Data Types

In compliance with the OSC standard, WING supports the following types:

- `int32` (32bits, bi-endian),
- `float32` (32bits, IEEE 754, big endian),
- `string` (non-null ASCII characters followed by a null, followed by 0-3 additional null characters to make the total number of bytes a multiple of 4),
- `blob` (An int32 size count, followed by one or more bytes of arbitrary binary data, followed by 0-3 additional zero bytes to make the total number of bytes a multiple of 4).

As specified in the `OSC` standard, the unit of transmission of `OSC` is an `OSC Packet`. Any application that sends `OSC Packets` is an `OSC Client`; WING embeds and runs an `OSC Server`.

An `OSC Packet` consists of its contents, a contiguous block of binary data, and its size, the number of 8-bit bytes that comprise the contents. The size of an `OSC packet` is always a multiple of 4.
In the case of WING, the contents of an `OSC packet` is always an `OSC Message`, i.e. `OSC Bundles` are not supported. Note that wildcards '`?`' and '`*`' in Address Patterns are reserved for special cases.
An `OSC Message` consists of an `OSC Address Pattern` followed by an `OSC Type Tag String` followed by zero or more `OSC Arguments`. Some older implementations of `OSC` may omit the `OSC Type Tag` string and WING supports this.

- `OSC Address Patterns` always start with the character '`/`'.
- `OSC Type Tags` can be `i, f, s, b` for int32, float32, string and blob, respectively
- `OSC Arguments` consist in a single or a contiguous sequence of the binary representations of each argument

The maximum `UDP` packet size is 32k bytes.

5 See http://opensoundcontrol.org/spec-1_0

WING OSC Messages

In the following paragraphs, we assume a communication link exists between WING and a client program, and communications take place with a WING console at a known `IP` address, using `UDP` on port `2223`.

All along this document, the character `'~'` will represent a `NULL` byte (`\0`). Patterns `->W` and `W->` represent data sent to WING and data received from WING followed by the actual number of bytes transmitted or received, respectively.

Retrieving WING console information can be completed by sending the OSC Address Pattern "`/?`"

```
->W,     4 B: /?~~
W->,    80 B: /?~~,s~~WING,192.168.1.71,PGM,ngc-full,NO_SERIAL,1.07.2-40-g1b1b292b:develop~~~~
```

The actual bytes exchanged are displayed below (OSC is a binary protocol)

```
->W,     4 B: 2f3f0000
W->,    80 B:
2f3f00002c73000057494e472c3139322e3136382e312e37312c50474d2c6e67632d66756c6c2c4e4f5f53455249414c2c
312e30372e322d34302d673162316232323932623a646576656c6f7000000000
```

The line below is using a more compliant OSC format, and will result in the same answer

```
->W,     8 B: /?~~,~~~
```

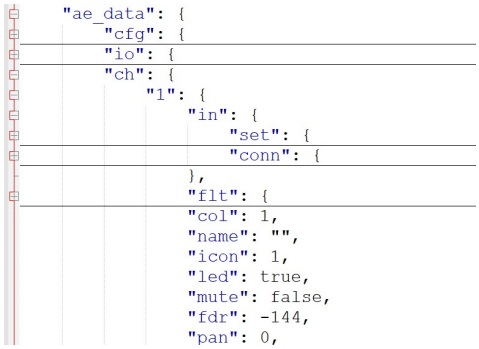
Reading (Get) Parameter and Node data

There are two main ways to gain access to WING data: using one-parameter-at-a-time or using "**nodes**".

WING "nodes" are a great way to access multiple parameters at a time, and therefore maximize communication bandwidth with the console. Nodes are represented as **string** OSC Data Type and are zero terminated (`\0` byte ending the string).
Nodes are also a good way to discover WING parameters, as they offer easy access to the full map of the `JSON` internal data structures.
We show below WING's first layer of `JSON` structure, and starting at the root, retrieved using OSC.

```
->W,     4 B: /~~~
W->,   116 B:
/~~~,sssssssssssssss~~~$stat~~~cfg~$syscfg~io~~ch~~aux~bus~main~~~~mtx~dca~mgrp~~~~fx~~cards~~~pl
ay~~~~rec~$ctl~~~~
```

Retrieving a WING single parameter is quite easy: You must ensure your OSC request points to a leaf of the `JSON` structure (i.e. there is no more hierarchy data after the current one). This is for example the case for the fader value of a channel strip, or its mute state. Channel Strip 1 fader is represented as follows:



Or "`ch`"/"`1`"/"`fdr`", which translates to OSC Address Pattern `/ch/1/fdr`:

```
->W,    12 B: /ch/1/fdr~~~
```

```
W->,   32 B: /ch/1/fdr~~~,sff~~~~-oo~[0.0000][-144.0000]
```

In the example above, the data `[0.0000][-144.0000]` are ascii representation of two 32bits big-endian float data values, each coded on 4 bytes as binary. The binary data actually received is as shown below, and in order to ease the reading of numerical information in this document, we use readable values in brackets rather than the actual binary data. The color highlights are there to help distinguish data elements.

```
W->,   32 B: 2f63682f312f666472000000 2c73666600000000 2d6f6f0000000000 c3100000
```

Depending on the OSC Address Pattern, WING returns '`,s`' for strings or enums, '`,sff`' (ascii, raw, float value) for floats, '`,sfi`' (ascii, raw, int value) for ints. In the example above, fader position is a float and WING returns the ascii representation, the raw `[0.0..1.0]` data and the actual float value in dB.
Similarly, requesting the mute state of channel strip 1 would return:

```
->W,   12 B: /ch/1/mute~~
W->,   32 B: /ch/1/mute~~,sfi~~~~1~~~[1.0000][     1]
W->,   32 B: 2f63682f312f6d7574650000 2c73666900000000 31000000 3f800000 00000001
```

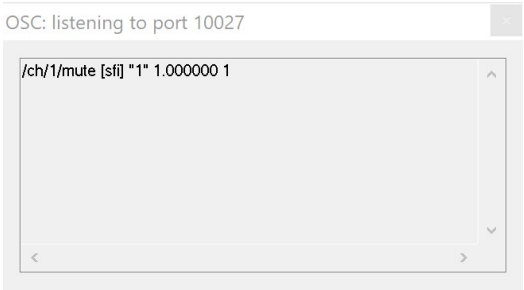
It should be noted that WING will accept both OSC path or the native hash data for representing nodes or parameters; Indeed, all nodes and parameters in the console are assigned a binary address (a hash) as explained in the chapter on native interface to the console. For example, the channel 1 mute command above can be sent as OSC Address Patterns `/ch/1/mute~~` as shown, or `/#f50f69f8~~`, and would return the same data as shown above. `0xf50f69f8` is the hash for command "Channel 1 mute". The full set of WING hash values can be discovered by recursively traversing the `JSON` tree of WING nodes/commands, using the native binary interface or OSC protocol, but it is generally more convenient to use the more standard OSC node notation, rather than hexadecimal hash values to address the console features.

Receiving OSC data on a specific port

Some OSC programs will request that data is returned on a specific port rather than being sent back to the port used by the requesting client for sending data. To enable this capability, WING OSC includes an optional, special notation for all OSC commands:
Any OSC command can be prefixed with the `/%<port>`, with `<port>` in the form "`12345`" to enable receiving the expected answer onto the specified port number. For example, the OSC request:

```
->W,   20 B: /%10027/ch/1/mute~~~
```

Will receive the expected reply from WING on `port 10027`, as shown below, using a sniffer program on said port. The `IP` does not change.



Writing (Set) Parameter and Node data

Single Parameters

OSC can be used to set or modify WING data. Taking the fader and mute examples above, we can modify their respective values using OSC commands, sending string, big-endian int32 or big-endian float32 with the corresponding OSC Type Tag following the OSC Address Pattern respective of the parameter to change.

WING does not echo data sent over UDP by the client application. The client application may nevertheless be notified with an OSC event in case of an error.
Individual parameters can be strings, integer, or floats; WING OSC server implementation enables to use several data types and will manage the conversion to ensure proper value setting inside the console. For example, fader position is a floating-point internal value. It can be set as a string or a float using the following OSC commands (in this example setting channel 2 fader position to -2 or -3dB):

```
->W,    20 B: /ch/2/fdr~~~,s~~-2~~
->W,    12 B: /ch/2/fdr~~~
W->,    36 B: /ch/2/fdr~~~,sff~~~~-2.0~~~~[0.7000][-2.0000]

->W,    20 B: /ch/2/fdr~~~,f~~[-3.0000]
->W,    12 B: /ch/2/fdr~~~
W->,    36 B: /ch/2/fdr~~~,sff~~~~-3.0~~~~[0.6750][-3.0000]
```

Enumerated strings

One of the data WING uses is "enumerated strings", or the choice of one string in a list of elements to represent a specific state or attribute value. For example, `/$ctl/user/1/1/enc/mode` can be any of the following strings: `OFF, FDR, PAN, DCA, SSND, FSND, FX, DAWMCU, MON, MIDICC, SD A`, or `SD B`

This can be set via a string OSC tag, as shown below if one wants to set the `mode` parameter to `FX`:

```
/$ctl/user/1/1/enc
->W,    20 B: /$ctl/user/1/1/enc~~
W->,    52 B: /$ctl/user/1/1/enc~~,sss~~~~mode~~~~name~~~~$fname~~
->W,    24 B: /$ctl/user/1/1/enc/mode~
W->,    32 B: /$ctl/user/1/1/enc/mode~,s~~OFF~
/$ctl/user/1/1/enc/mode ,s FX
->W,    32 B: /$ctl/user/1/1/enc/mode~,s~~FX~~
/$ctl/user/1/1/enc/mode
->W,    24 B: /$ctl/user/1/1/enc/mode~
W->,    32 B: /$ctl/user/1/1/enc/mode~,s~~FX~~
```

But it can also be set as an int OSC tag, using the index of the list corresponding to the targeted value; in the example above, `FX` sits at index `6` in the list of `10` strings; This enables us to use the following OSC command to set the encoder mode to `FX`:

```
/$ctl/user/1/1/enc
->W,    20 B: /$ctl/user/1/1/enc~~
W->,    52 B: /$ctl/user/1/1/enc~~,sss~~~~mode~~~~name~~~~$fname~~
->W,    24 B: /$ctl/user/1/1/enc/mode~
W->,    32 B: /$ctl/user/1/1/enc/mode~,s~~OFF~
/$ctl/user/1/1/enc/mode ,i 6
->W,    32 B: /$ctl/user/1/1/enc/mode~,i~~[     6]
/$ctl/user/1/1/enc/mode
->W,    24 B: /$ctl/user/1/1/enc/mode~
W->,    32 B: /$ctl/user/1/1/enc/mode~,s~~FX~~
```

One can also note the extendibility character of WING nodes; indeed, after the previous command, the user 1/1 encoder has additional parameters:

```
/$ctl/user/1/1/enc
->W,    20 B: /$ctl/user/1/1/enc~~
W->,    60 B: /$ctl/user/1/1/enc~~,sssss~~mode~~~~name~~~~$fname~~fx~~par~
```

Node Data

WING nodes can also be used to set multiple values with using a single OSC "`/`" command, and offer a simple yet effective way to navigate within the hierarchical structure of `JSON` data. Say you want/need to set fader and mute values to -1 dB, 0 dB, `OFF` and `ON` for channels 1 and 2; This can be achieved in a single OSC request using the following syntax:

```
->W,    44 B: /~~~,s~~/ch.1.fdr=-1,mute=0,.2.fdr=0,mute=1~
```

Or setting channel 1 fader and mute values to 10 dB and ON, and setting bus 1 fader to 5 dB:

```
->W,   44 B: /~~~,s~~/ch.1.fdr=10,mute=1,/bus.1.fdr=5~~~~
```

As shown above, each parameter group is separated by a '`,`' character, the '`/`' character represents the root of the `JSON` parameter tree, and '`.`' characters are used to navigate up and down within the `JSON` parameter tree. The console will reply with `/*~~,s~~OK~~` if the command was accepted, or one of the following:

```
/*~~,s~~NODE NOT FOUND~~
/*~~,s~~VALUE ERROR~~~~~
/*~~,s~~BUFFER OVERFLOW~
/*~~,s~~NODE IS NOT PAR~
/*~~,s~~INCOMPLETE DATA~
/*~~,s~~STACK EMPTY~~~~~
```

if an error occurred during the execution of the command.

Note: Nodes can return large amounts of data; as a result, some nodes cannot be returned using `OSC/UDP` as they would overflow the `32kB UDP` buffer limitation; In such situation, WING will return an error OSC message event.

Some nodes examples are provided below:

```
->W,   12 B: /ch/1/fdr~~~
W->,   32 B: /ch/1/fdr~~~,sff~~~~-oo~[0.0000][-144.0000]
->W,   12 B: /ch/1/mute~~
W->,   32 B: /ch/1/mute~~,sfi~~~~1~~~[1.0000][     1]
->W,   12 B: /ch/2/fdr~~~
W->,   32 B: /ch/2/fdr~~~,sff~~~~-oo~[0.0000][-144.0000]
->W,   12 B: /ch/2/mute~~
W->,   32 B: /ch/2/mute~~,sfi~~~~0~~~[0.0000][     0]

->W,   44 B: /~~~,s~~/ch.1.fdr=-1,mute=0,.2.fdr=0,mute=1~
W->,   12 B: /*~~,s~~OK~~

->W,   12 B: /ch/1/fdr~~~
W->,   36 B: /ch/1/fdr~~~,sff~~~~-1.0~~~~[0.7250][-1.0000]
->W,   12 B: /ch/1/mute~~
W->,   32 B: /ch/1/mute~~,sfi~~~~0~~~[0.0000][     0]
->W,   12 B: /ch/2/fdr~~~
W->,   32 B: /ch/2/fdr~~~,sff~~~~0.0~[0.7500][0.0000]
->W,   12 B: /ch/2/mute~~
W->,   32 B: /ch/2/mute~~,sfi~~~~1~~~[1.0000][     1]
```

Nodes can also be located deeper in the `JSON` structure tree. For example, changing a single parameter in the node channel 1 ["`/ch/1`"] can be done as shown below:

```
->W,   20 B: /ch/1~~~,s~~fdr=3~~~
W->,   16 B: /ch/1*~~,s~~OK~~

->W,   12 B: /ch/1/fdr~~~
W->,   32 B: /ch/1/fdr~~~,sff~~~~3.0~[0.8250][3.0000]
->W,   12 B: /ch/1/mute~~
W->,   32 B: /ch/1/mute~~,sfi~~~~0~~~[0.0000][     0]
```

The OSC command is replied to with an OK status if execution went well; error messages can be returned too, as explained earlier.

The same type of command can be used to set/change several parameters at once; For example, fader and mute values of channel 1 can be done as follows:

```
->W,   28 B: /ch/1~~~,s~~fdr=4,mute=1~~~~
W->,   16 B: /ch/1*~~,s~~OK~~

->W,   12 B: /ch/1/fdr~~~
W->,   32 B: /ch/1/fdr~~~,sff~~~~4.0~[0.8500][4.0000]
->W,   12 B: /ch/1/mute~~
W->,   32 B: /ch/1/mute~~,sfi~~~~1~~~[1.0000][     1]
```

Special Node Type/Arguments

There are three special tag/argument that are specifically implemented for nodes. They enable listing the complete set of data, parameter description, and description including values for the node provided as OSC address pattern. The arguments to use are '`*`', '`?`', and '`#`', respectively. Examples of use are provided below, applied to OSC address pattern `/fx/1` when no effect is loaded to keep the description as short as possible.

Node data dump:

When using this format, The data returned will strictly correspond to what would be saved in a snap file; Read-only and temporary data are not returned.

```
/fx/1 ,s *
->W,   16 B: /fx/1~~~,s~~*~~~
W->,   32 B: /fx/1~~~,s~~mdl=NONE,fxmix=100,~
```

Node parameter description:

```
/fx/1 ,s ?
->W,   16 B: /fx/1~~~,s~~?~~~
W->,  696 B: /fx/1~~~,s~~  mdl            list [NONE, EXT, HALL, ROOM, CHAMBER, PLATE, CONCERT,
AMBI, V-ROOM, V-REV, V-PLATE, GATED, REVERSE, DEL/REV, SHIMMER, SPRING, DIMCRS, CHORUS, FLANGER,
ST-DL, TAP-DL, TAPE-DL, OILCAN, BBD-DL, PITCH, D-PITCH, VSS3, BPLATE, GEQ, PIA, DOUBLE, PCORR,
LIMITER, DE-S2, ENHANCE, EXCITER, P-BASS, ROTARY, PHASER, PANNER, TAPE, MOOD, SUB, RACKAMP,
UKROCK, ANGEL, JAZZC, DELUXE, BODY, SOUL, E88, E84, F110, PULSAR, MACH4, C5-CMB, SUB-M, V-IMG,
SPKMAN, DEQ3, *EVEN*, *SOUL*, *VINTAGE*, *BUS*, *MASTER*]~  fxmix          lin [0 .. 100 %], 101
steps~  $esrc          int [0 .. 400]~  $emode         list [M, ST, M/S]~  $a_chn         int [0
.. 76]~  $a_pos         int [0 .. 1]~~~~
```

Node description including values:

```
/fx/1 ,s #
->W,   16 B: /fx/1~~~,s~~#~~~
W->,  816 B: /fx/1~~~,s~~  mdl            NONE                list [NONE, EXT, HALL, ROOM,
CHAMBER, PLATE, CONCERT, AMBI, V-ROOM, V-REV, V-PLATE, GATED, REVERSE, DEL/REV, SHIMMER, SPRING,
DIMCRS, CHORUS, FLANGER, ST-DL, TAP-DL, TAPE-DL, OILCAN, BBD-DL, PITCH, D-PITCH, VSS3, BPLATE,
GEQ, PIA, DOUBLE, PCORR, LIMITER, DE-S2, ENHANCE, EXCITER, P-BASS, ROTARY, PHASER, PANNER, TAPE,
MOOD, SUB, RACKAMP, UKROCK, ANGEL, JAZZC, DELUXE, BODY, SOUL, E88, E84, F110, PULSAR, MACH4,
C5-CMB, SUB-M, V-IMG, SPKMAN, DEQ3, *EVEN*, *SOUL*, *VINTAGE*, *BUS*, *MASTER*]~  fxmix
100                lin [0 .. 100 %], 101 steps~  $esrc          0               r/o int [0 ..
400]~  $emode         M               r/o list [M, ST, M/S]~  $a_chn         0               r/o
int [0 .. 76]~  $a_pos         0               r/o int [0 .. 1]~~~~
```

As a second example, we give below the node data dump for OSC address pattern `/ch/1`, when loaded with default values after init:

```
/ch/1 ,s *
->W,   16 B: /ch/1~~~,s~~*~~~
W->, 1972 B:
/ch/1~~~,s~~in.set.srcauto=0,altsrc=0,inv=0,trim=0.0,bal=0.0,dlymode=M,dly=0.0,dlyon=0,.conn.grp=L
CL,in=1,altgrp=OFF,altin=1,..flt.lc=0,lcf=100.2,hc=0,hcf=10k02,tf=0,mdl=TILT,tilt=0.00,.clink=0,co
l=1,name=,icon=1,led=1,mute=0,fdr=-oo,pan=0,wid=100,solosafe=0,mon=A,proc=GEDI,ptap=5,peq.on=0,1g=
0.0,1f=100,1q=1.00,2g=0.0,2f=999,2q=1.00,3g=0.0,3f=10k0,3q=1.00,.gate.on=0,mdl=GATE,thr=-40.0,rang
e=40.0,att=10,hld=10,rel=199,acc=0,ratio='1:3',.gatesc.type=OFF,f=1k0,q=2.00,src=SELF,tap=IN,.eq.o
n=0,mdl=STD,mix=100,lg=0.0,lf=80.2,lq=1.00,leq=SHV,1g=0.0,1f=200.0,1q=1.00,2g=0.0,2f=601.4,2q=1.00
,3g=0.0,3f=1k50,3q=1.00,4g=0.0,4f=3k99,4q=1.00,hg=0.0,hf=11k99,hq=1.00,heq=SHV,.dyn.on=0,mdl=COMP,
```

mix=100,gain=0.0,thr=-10.0,ratio=3.0,knee=3,det=RMS,att=50,hld=20,rel=153,env=LOG,auto=1,.dynxo.depth=6.0,type=OFF,f=1k0,.dynsc.type=OFF,f=1k0,q=2.00,src=SELF,tap=IN,.preins.on=0,ins=NONE,.main.1.on=1,lvl=0.0,pre=0,.2.on=0,lvl=0.0,pre=0,.3.on=0,lvl=0.0,pre=0,.4.on=0,lvl=0.0,pre=0,..send.1.on=0,lvl=-oo,pon=0,ind=0,mode=PRE,plink=0,pan=0,wid=100,.2.on=0,lvl=-oo,pon=0,ind=0,mode=PRE,plink=0,pan=0,wid=100,.3.on=0,lvl=-oo,pon=0,ind=0,mode=PRE,plink=0,pan=0,wid=100,.4.on=0,lvl=-oo,pon=0,ind=0,mode=PRE,plink=0,pan=0,wid=100,.5.on=0,lvl=-oo,pon=0,ind=0,mode=PRE,plink=0,pan=0,wid=100,.6.on=0,lvl=-oo,pon=0,ind=0,mode=PRE,plink=0,pan=0,wid=100,.7.on=0,lvl=-oo,pon=0,ind=0,mode=PRE,plink=0,pan=0,wid=100,.8.on=0,lvl=-oo,pon=0,ind=0,mode=PRE,plink=0,pan=0,wid=100,.9.on=0,lvl=-oo,pon=0,ind=0,mode=PRE,plink=0,pan=0,wid=100,.10.on=0,lvl=-oo,pon=0,ind=0,mode=PRE,plink=0,pan=0,wid=100,.11.on=0,lvl=-oo,pon=0,ind=0,mode=PRE,plink=0,pan=0,wid=100,.12.on=0,lvl=-oo,pon=0,ind=0,mode=PRE,plink=0,pan=0,wid=100,.13.on=0,lvl=-oo,pon=0,ind=0,mode=PRE,plink=0,pan=0,wid=100,.14.on=0,lvl=-oo,pon=0,ind=0,mode=PRE,plink=0,pan=0,wid=100,.15.on=0,lvl=-oo,pon=0,ind=0,mode=PRE,plink=0,pan=0,wid=100,.16.on=0,lvl=-oo,pon=0,ind=0,mode=PRE,plink=0,pan=0,wid=100,..postins.on=0,mode=FX,ins=NONE,w=0.0,.tags=,~~~

OSC: Special Cases

JSON Structure dynamic changes

As parameters get changed on the WING console, its `JSON` structure tree evolves to reflect the changes; This can be a specific parameter that when changing to an `ON` state, offers new capabilities in the audio chain, or in the way the console will react.
It is also typical of **effects** and **plugins**: WING consoles support dynamic allocation of effect or plugins that can generate large changes within the default `JSON` tree. As already mentioned, WING nodes are a great way to list the parameters available for a given effect and therefore a way to get and possibly set effect parameter values.
The WING effects and plugins, and their respective parameters are listed later in this document[6].

The OSC commands below show how you can access effects slots, allocate an effect, and list parameters and later modify effect parameter values.

Accessing effects with currently no effect loaded in effect slot 1, listing the effect node:

```
->W,     4 B: /fx~
W->,    88 B:
/fx~,sssssssssssssss~~~1~~~2~~~3~~~4~~~5~~~6~~~7~~~8~~~9~~~10~~11~~12~~13~~14~~15~~16~~

->W,     8 B: /fx/1~~~
W->,    60 B: /fx/1~~~,ssssss~mdl~fxmix~~~$esrc~~~$emode~~$a_chn~~$a_pos~~
->W,    12 B: /fx/1/mdl~~~
W->,    24 B: /fx/1/mdl~~~,s~~NONE~~~~
```

Loading a PIA effect in effect slot 1:

```
->W,    20 B: /fx/1/mdl~~~,s~~pia~
->W,    12 B: /fx/1/mdl~~~
W->,    20 B: /fx/1/mdl~~~,s~~PIA~
```

PIA effect is now loaded, listing the effect Node gives a different set of parameters:

```
->W,     8 B: /fx/1~~~
W->,   120 B:
/fx/1~~~,sssssssssssssssss~mdl~fxmix~~~$esrc~~~$emode~~$a_chn~~$a_pos~~mix~g~~~31~~63~~125~250~50
0~1k~~2k~~4k~~8k~~16k~
```

We can now get/set effect 1 PIA parameters, for example the 125Hz band:

```
->W,    12 B: /fx/1/125~~~
W->,    32 B: /fx/1/125~~~,sff~~~~0.0~[0.5000][0.0000]
```

The 125Hz band is at 0dB, change it to 10dB and verify the change:

```
->W,    20 B: /fx/1/125~~~,f~~[10.000]
->W,    12 B: /fx/1/125~~~
W->,    36 B: /fx/1/125~~~,sff~~~~10.0~~~~[0.9233][10.000]
```

[6] Please refer to the "Effects" paragraph

OSC Tag Type 'blob' or 'binary' use

WING OSC server implementation supports the 'blob'/'binary' OSC Tag type, enabling the use of 'native' commands[7] within OSC, making it is possible with the proper information at hand to send and receive binary data.

An alternative to standard node requests (such as the request on root below) is to use binary.

```
->W,    4 B: /~~~
W->,  116 B:
/~~~,sssssssssssssss~~~$stat~~~cfg~$syscfg~io~~ch~~aux~bus~main~~~~mtx~dca~mgrp~~~~fx~~cards~~~play~~~~rec~$ctl~~~~
```

Binary types typically apply on WING nodes to retrieve the internal binary equivalent of the `JSON` tree level respective of a WING node.
Shown below is a request at root level using the native commands part of the binary data [all bytes sent shown as hex data]

```
/ ,b dd
```

Data actually sent (in hex): `->W,    16 B: 2f0000002c620000000000001dd000000`
WING's reply is:

```
W->,  440 B: /~~~,b~~425 bytes:
df00180000000097a00439000005247374617405535441544500 00df001100000000edca7af900000363666700000 0df00
1500000000f89818a60000072473797363666700000 0df00130000000294f7794000002696f03492f4f0000df00170000
0000070b10139000002636807434841 4e4e454c0000df001c000000008fa3078d0000036175780b415558204348414e4e45
4c0000df001400000000f46c185e000003627573034255530000df00160000000004d3a3a80000046d61696e044d41494e
0000df001700000000f82a5af20000036d7478064d41545249580000df001400000000e313aeff000003646361034443341
0000df001c00000000d252398b0000046d6772700a4d5554452047524f55500000df001700000000473c91340000026678
074546464543545300 00df002200000000b4296fc900000563617264730f455850414e53494f4e2043415244530000df00
180000000057297a28000004706c617906504c415945520000df001900000000fab1762c000003726563085245434f5244
45520000df001900000000cbb9514300000424637 46c07434f4e54524f4c0000de
```

Lots of information are returned either as string, or more often as blob/binary. In the reply above, after each '`df`' byte is a data length on two bytes, immediately followed by the binary address (the hash) where a node, parameter, or subtree data can be found. For example, the subtree entry for channel (`/ch`) can be found at address/hash `70b10139`

An example on retrieving the DAW node (hash is `df17c242`, part of the `$ctl` subtree) is shown below. Sending the OSC blob:

```
/$ctl/daw ,b dd
```
or
```
/ ,b d7df17c242dd
```

Respectively translate in the following binary data being sent to the console:

```
->W,  24 B: 2f2463746c2f6461770000002c6200000000001dd000000
or
->W,  20 B: 2f0000002c62000000000006d7df17c242dd0000
```

To which the console replies with (it can also reply with one of the errors listed earlier in the OSC chapters):

```
W->,  876 B: /$ctl/daw~~~,b~~856 bytes:
df0022df17c2423cb129d50000026f6e0a44415720454e41424c450040000000000000001df0028df17c2424e5c7f3400
0004636f6e6e0a434f4e4e454354494f4e005000020344494e000355534200df0027df17c242e5681680000004656d756c
09454d554c4154494f4e00500002034d435500034855 4900df0071df17c24242701ca9000006636f6e6669670000500004
0243431443555354 4f4d20434f4e54524f4c53204f4e4c59044d5354520a53494e474c45204d4355084d53545231455854
0e4d4355202b20455854454e444552084d53545232455854114d4355202b20327820455854454e444552df002edf17c242
ae1538a400000463637570145553452055505045522043432046 4f52204441570040000000000000001df0035df17c242
9fa4e732000006646973 6a6f671944495341424c4520574845454c20445552494e4720504c41590040000000000000001
```

[7] Detail information on native commands is provided in a separate chapter

```
df0097df17c242892e512d0000067072657365741 24c415354204c4f414445442050524553455400500008012d012d0663
756261736506435542415345046c697665044c495645066c6f67696378074c4f4749432058066e75656e646f064e55454e
444f0870726f746f6f6c730950524f20544f4f4c53067265617065720652454150455209737475646 96f6f6e650a535455
44494f204f4e45df001fdf17c242beefaeab000003246f6e06444157204f4e024000000000000000001df0027df17c24296
31559f0000062462706167650b425554544f4e2050414745004000000000000000004df0031df17c242012dc54600000924
62746e746f7563681242544e53454c20464144455220544f554348004000000000000000001df002adf17c242775c19c200
00082462746e76706f740c42544e53454c20562d504f5400400000000000000001df002ddf17c24242aeb92800000a2462
746e7265637264790d42544e53454c2052454352445900400000000000000001df0029df17c242fccfbe07000008246274
6e6175746f0b42544e53454c204155544f00400000000000000001df002adf17c24285cdce3f0000082462746e7673656c
0c42544e53454c20562d53454c00400000000000000001df002ddf17c24215abd96800000a2462746e696e736572740d42
544e53454c20494e534552540040000000000000001de
```

The above is more difficult to read than the more standard way of retrieving the node, but contains more information:

```
->W,  12 B: /$ctl/daw~~~
W->, 156 B:
/$ctl/daw~~~,sssssssssssss~on~~conn~~~~emul~~~~config~~ccup~~~~preset~~$on~$bpage~~$btntouch~~~$b
tnvpot~~~~$btnrecrdy~~$btnauto~~~~$btnvsel~~~~$btninsert~~
```

Matching the two representations tell us that:
`daw/on` is at binary address `3cb129d5`,
`daw/conn` at `4e5c7f34`,
`daw/emul` at `e5681680`,
`daw/config` at `42701ca9`,
`daw/ccup` at `ae1538a4`,
`daw/preset` at `892e512d`,
`daw/$on` at `beefaeab`,
and so on (highlighted values above).

The blob /binary Type Tag can also be used to execute native/binary commands. Using for example the `daw/$on` hash/binary address value of `beefaeab`, we can set the console in and out of DAW mode, as if one would have pressed the DAW button.
For example, sending any of the following commands will set DAW mode ON:

```
/ ,b d7beefaeab01
->W,   20 B: /~~~,b~~6 bytes: d7beefaeab01
W->,   12 B: /*~~,s~~OK~~
/$ctl/daw/$on ,b 01
->W,   28 B: /$ctl/daw/$on~~~,b~~1 bytes: 01~~~
W->,   12 B: /*~~,s~~OK~~
```

In the binary data sent with the line above, the segment `01` is equivalent to asking the value of the parameter to be set using a 32bit integer with value 1.

The following lines are requesting to turn OFF DAW mode:

```
/ ,b d7beefaeab00
->W,   20 B: /~~~,b~~6 bytes: d7beefaeab00
W->,   12 B: /*~~,s~~OK~~
/$ctl/daw/$on ,b 00
->W,   28 B: /$ctl/daw/$on~~~,b~~1 bytes: 00~~~
W->,   12 B: /*~~,s~~OK~~
```

In both blob Type Tag commands above, the console replies with a blob. Depending on the cases, it can also return strings.

As seen above, the Tag Type blob can be used to retrieve the description of WING parameters when using the native command 'data description' a.k.a. '`dd`'; In an example below, still using the DAW ON state, we can get the data using the following command:

```
/$ctl/daw/$on ,b dd
->W,   28 B: /$ctl/daw/$on~~~,b~~1 bytes: dd~~~
```

WING returns the following which includes the hash value for `/$ctl/daw/$on` and its full description:

```
W->,   60 B: /$ctl/daw/$on~~~,b~~35 bytes:
df001fdf17c242beefaeab0000003246f6e06444157204f4e0040000000000000001de
parse 35 bytes node
    len: 31, parent: df17c242, hash: beefaeab, index: 0, flags: 0040
    name: $on longname: DAW ON, type: <int> [0..1]

End node
```

The blob Tag Type can be used to retrieve the value of WING parameters when using the native command 'data request', a.k.a. **'dc'**; In an example below, still using the DAW ON state, we can get the data using the following command:

```
->W,   20 B: /~~~,b~~6 bytes: d7beefaeabdc
W->,   20 B: /~~~,b~~7 bytes: d7beefaeab01de

With 01 indicating the DAW [Remote control] button is in an ON state.
```

Detailed information on the native/binary interface to WING and data value coding is provided later in this document.

Subscribing to OSC Data

There are three main types of subscription for receiving binary or OSC messages.
A single OSC subscription is active at any time, provided to the last requestor. Subscriptions must be renewed every `10 seconds` to keep the subscription alive by sending one of the 3 messages shown below.

`/*b~` (or `/*b~,~~~`) will enable receiving event driven binary messages
Binary messages are formatted exactly as the binary/native interface and therefore can be sent back to the console with no change.

Example using mutes and faders

```
->W,    4 B: /*b~
W->,   32 B: /~~~,b~~20 bytes: d738ae75c2d5c3100000d77e463474d5c3100000
W->,   24 B: /~~~,b~~12 bytes: d7f50f69f801d726855cd301
```

`/*s~` (or `/*s~,~~~`) will enable receiving event OSC messages
OSC messages are received as triplets of data, as previously presented[8], and shown below; Sending back data to WING will require to select one of the (up to) 3 parameters received, depending on the chosen format. The 'string' argument will always work for all messages.

Example using mutes and faders

```
->W,    4 B: /*s~
W->,   32 B: /ch/1/fdr~~~,sff~~~~-oo~[0.0000][-144.0000]
W->,   32 B: /ch/1/$fdr~~,sff~~~~-oo~[0.0000][-144.0000]
W->,   32 B: /ch/1/mute~~,sfi~~~~1~~~[1.0000][     1]
W->,   32 B: /ch/1/$mute~,sfi~~~~1~~~[0.5000][     1]
```

`/*S~` (or `/*S~,~~~`) will enable receiving event OSC messages
OSC messages are received as single tag data, as shown below; WING reports the native format of the OSC pattern (ex: '`f`' for floats, '`i`' for integers, etc.). Data received with events resulting of a `/*S~` subscription can be sent back to the console with no change.

Example using mutes and faders

```
->W,    4 B: /*S~
W->,   20 B: /ch/1/fdr~~~,f~~[-144.0000]
W->,   20 B: /ch/1/$fdr~~,f~~[-144.0000]
W->,   20 B: /ch/1/mute~~,i~~[     1]
W->,   20 B: /ch/1/$mute~,i~~[     1]
```

Using the simple forms of subscription requests will provide data from the console to the requesting IP/port. It is possible to redirect the data received from WING by prefixing the commands with a port specifier element as shown below:

`/%23456/*b~` will subscribe to binary messages, being sent by WING to port `23456`.
`/%23456/*s~` will subscribe to OSC messages, being sent by WING to port `23456`.
`/%23456/*S~` will subscribe to OSC messages, being sent by WING to port `23456`.

[8] Refer to "Writing (Set) Parameter and Node data", paragraph "Single Parameters"

WING ae_data OSC commands list

The next chapters provide an abridged[9] list of all OSC commands available for WING.

All commands and parameters below are part of the ***ae_data*** section in `JSON` snapshot files. Other console control commands part of the ***ce_data*** section in `JSON` snapshot files are described later in this document.

Status

Command	Type	Range	Text	Description
/$stat	N			Status node
/$stat/A	N			AES50 A node
/$stat/A/stat	S		-, OK, ERR, UPD	AES50 A state [RO]
/$stat/A/dev	S		128 chars max	AES50 A Device [RO]
/$stat/B	N			AES50 B node
/$stat/B/stat	S		-, OK, ERR, UPD	AES50 B state [RO]
/$stat/B/dev	S		128 chars max	AES50 B Device [RO]
/$stat/C	N			AES50 C node
/$stat/C/stat	S		-, OK, ERR, UPD	AES50 C state [RO]
/$stat/C/dev	S		128 chars max	AES50 C Device [RO]
/$stat/lock	I	0..1		Clock lock [RO]
/$stat/ppm	I	-200..200		Clock ppm [RO]
/$stat/solo	I	0..1		Solo [RO]
/$stat/sip	I	0..1		Solo In Place [RO]
/$stat/rtcerr	I	0..1		Real Time Clock Error [RO]
/$stat/time	S		12 chars max	Clock time (depending on time format) [RO]
/$stat/date	S		12 chars max	Clock date (depending on date format) [RO]
/$stat/usbstate	S		-, ERR, IDLE, BUSY	USB Player state [RO]
/$stat/usbvolname	S		20 chars max	USB Player volume name [RO]
/$stat/sc_stat	S		OK, ERR	StageConnect status [RO]
/$stat/sc_devices	S		128 chars max	StageConnect devices [RO]
/$stat/sc_upcnt	I	0..32		StageConnect upstreams [RO]
/$stat/sc_dncnt	I	0..32		StageConnect downstreams [RO]
/$stat/sc_uprout	S		32 char max	StageConnect upstream routing [RO]
/$stat/rmt_a	S		16 chars max	Name of the console connected on AES50 port A [RO]
/$stat/rmt_b	S		16 chars max	Name of the console connected on AES50 port A [RO]
/$stat/rmt_c	S		16 chars max	Name of the console connected on AES50 port A [RO]

[9] It includes the set of commands for the first element of a series. For example, `/ch/1` set of OSC commands are listed, but not `/ch/2` to `/ch/40`.

General Configuration

Command	Type	Range	Text	Description
/cfg	N			General Configuration node
/cfg/mainlink	S		OFF, 2, 2-3, 2-4	Main Link
/cfg/dcamgrp	I	0..1		DCA mutegroups (DCA mute mutes all channels assigned to DCA)
/cfg/mon	N			Monitor buses config node
/cfg/mon/1	N	1..2		Monitor bus 1 node
/cfg/mon/1/$lvl	F	-144..10	-oo..10 in 1024 steps	Monitor bus 1 level (dB)[10]
/cfg/mon/1/inv	I	0..1		Monitor bus 1 invert (polarity)
/cfg/mon/1/pan	F	-100..100	201 steps	Monitor bus 1 pan
/cfg/mon/1/wid	F	-150..150	61 steps	Monitor bus 1 width (%)
/cfg/mon/1/eq	N			Monitor bus 1 EQ node
/cfg/mon/1/eq/on	I	0..1		Monitor bus 1 EQ off/on
/cfg/mon/1/eq/lsg	F	-15..15	301 steps	Monitor bus 1 EQ low shelf gain (dB)
/cfg/mon/1/eq/lsf	F	20..2000	641 steps	Monitor bus 1 EQ low shelf frequency (Hz)
/cfg/mon/1/eq/1g	F	-15..15	301 steps	Monitor bus 1 EQ band 1 gain (dB)
/cfg/mon/1/eq/1f	F	20..20000	961 steps	Monitor bus 1 EQ band 1 frequency (Hz)
/cfg/mon/1/eq/1q	F	0.44..10	181 steps	Monitor bus 1 EQ band 1 Q
/cfg/mon/1/eq/2g	F	-15..15	301 steps	Monitor bus 1 EQ band 2 gain (dB)
/cfg/mon/1/eq/2f	F	20..20000	961 steps	Monitor bus 1 EQ band 2 frequency (Hz)
/cfg/mon/1/eq/2q	F	0.44..10	181 steps	Monitor bus 1 EQ band 2 Q
/cfg/mon/1/eq/3g	F	-15..15	301 steps	Monitor bus 1 EQ band 3 gain (dB)
/cfg/mon/1/eq/3f	F	20..20000	961 steps	Monitor bus 1 EQ band 3 frequency (Hz)
/cfg/mon/1/eq/3q	F	0.44..10	181 steps	Monitor bus 1 EQ band 3 Q
/cfg/mon/1/eq/4g	F	-15..15	301 steps	Monitor bus 1 EQ band 4 gain (dB)
/cfg/mon/1/eq/4f	F	20..20000	961 steps	Monitor bus 1 EQ band 4 frequency (Hz)
/cfg/mon/1/eq/4q	F	0.44..10	181 steps	Monitor bus 1 EQ band 4 Q
/cfg/mon/1/eq/5g	F	-15..15	301 steps	Monitor bus 1 EQ band 5 gain (dB)
/cfg/mon/1/eq/5f	F	20..20000	961 steps	Monitor bus 1 EQ band 5 frequency (Hz)
/cfg/mon/1/eq/5q	F	0.44..10	181 steps	Monitor bus 1 EQ band 5 Q
/cfg/mon/1/eq/6g	F	-15..15	301 steps	Monitor bus 1 EQ band 6 gain (dB)
/cfg/mon/1/eq/6f	F	20..20000	961 steps	Monitor bus 1 EQ band 6 frequency (Hz)
/cfg/mon/1/eq/6q	F	0.44..10	181 steps	Monitor bus 1 EQ band 6 Q
/cfg/mon/1/eq/hsg	F	-15..15	301 steps	Monitor bus 1 EQ high shelf gain (dB)
/cfg/mon/1/eq/hsf	F	50..20000	833 steps	Monitor bus 1 EQ high shelf frequency (Hz)
/cfg/mon/1/lim	F	-40..0	41 steps	Monitor bus 1 limiter level(dB)
/cfg/mon/1/dly	N			Monitor bus 1 delay node
/cfg/mon/1/dly/on	I	0..1		Monitor bus 1 delay off/on

[10] This command is considered RO on the full-size WING, and can be set for other devices where the actual surface control potentiometer is not present.

/cfg/mon/1/dly/m	F	0.1..100	1000 steps	Monitor bus 1 delay (meters)
/cfg/mon/1/dim	F	40..0	41 steps	Monitor bus 1 delay dim level (dB)
/cfg/mon/1/pfldim	F	40..0	41 steps	Monitor bus 1 PFL Dim (dB)
/cfg/mon/1/eqbdtrim	F	0..24	25 steps	Monitor bus 1 band solo trim {dB)
/cfg/mon/1/srclvl	F	-144..10	-oo..10 in 1024 steps	Monitor bus 1 source level
/cfg/mon/1/srcmix	F	-144..10	-oo..10 in 1024 steps	Monitor bus 1 source mix (dB)
/cfg/mon/1/src	S		OFF, MAIN.1..MAIN.4, MTX.1..MTX.8, BUS.1..BUS.16, AUX.1..AUX.8	Monitor bus 1 source
/cfg/mon/1/$lvlact	F	-144..10	-oo..10 in 1024 steps	Monitor bus 1 fader level [RO]
/cfg/mon/1/tags	S		Up to 80 chars	Monitor bus 1 tags
/cfg/solo	N			Solo config node
/cfg/solo/mode	S		LIVE, STUDIO, SIP	Solo mode
/cfg/solo/mon	S		A, B, A+B	Solo monitor
/cfg/solo/mute	I	0..1		Solo mute
/cfg/solo/$dim	I	0..1		Solo dim off/on
/cfg/solo/$mono	I	0..1		Solo mono off/on
/cfg/solo/$flip	I	0..1		Solo left and right channels flipped
/cfg/solo/chtap	S		PFL, AFL	Solo channel tap
/cfg/solo/bustap	S		PFL, AFL	Solo bus tap
/cfg/solo/maintap	S		PFL, AFL	Solo main tap
/cfg/solo/mtxtap	S		PFL, AFL	Solo matrix tap
/cfg/solo/srcsolo	S		OFF, CH39, AUX7	Source Solo Enable
/cfg/solo/$srcsolo	I	0..1		Source Solo
/cfg/solo/$srcsgrp	I	1..13		Source Solo Group
/cfg/solo/$srcsin	I	1..64		Source Solo In
/cfg/rta[11] [12]	N			RTA config node (dsp)
/cfg/rta/$src	I	1..76		RTA source [RO]
/cfg/rta/$tap	S		IN, POST, FILT, PREEQ, POSTEQ, PREFDR, GATEK, DYNK, DYNXO, PRETAP, SOLO, MON.A, MON.B, FXIN, FXOUT	RTA source tap [RO]
/cfg/rta/$dec	S		SLOW, MED, FAST	RTA Decay [RO]
/cfg/rta/$det	S		PEAK, RMS	RTA Detector [RO]
/cfg/rta/rtasrc	I	0..76		*RTA source (indexed)
/cfg/rta/rtatap	S		IN, POST, FILT, PREEQ, POSTEQ, PREFDR, GATEK, DYNK, DYNXO, PRETAP, SOLO, MON.A, MON.B	*RTA source tap
/cfg/rta/rtadecay	S		SLOW, MED, FAST	*RTA decay
/cfg/rta/rtadet	S		PEAK, RMS, AVG	*RTA detector
/cfg/rta/rtarange	F	30, 60		*RTA range (dB)
/cfg/rta/rtagain	F	-5..50	56 steps	*RTA gain (dB)
/cfg/rta/rtaauto	I	0..1		*RTA autogain
/cfg/rta/eqdecay	S		SLOW, MED, FAST	RTA eq decay
/cfg/rta/eqdet	S		PEAK, RMS, AVG	RTA eq detector
/cfg/rta/eqrange	F	30, 60		RTA eq range (dB)
/cfg/rta/eqgain	F	-5..50	56 steps	RTA eq gain (dB)
/cfg/rta/eqauto	I	0..1		RTA eq autogain
/cfg/mtr	N			Meter config node

[11] Tags (marqued with *) are only used with metering RTA and future RTA screen, as opposed to EQ (on-screen) RTA

[12] See also /$ctl/cfg/rta commands

/cfg/mtr/$scopesrc	I	1..76		Meter scope source [RO]
/cfg/mtr/$scopetap	S		IN, POST, FILT, PREEQ, POSTEQ, PREFDR, GATEK, DYNK, DYNXO, PRETAP, SOLO, MON.A, MON.B, FXIN, FXOUT	Meter scope source tap point [RO]
/cfg/mtr/scopesrc	I	0..76		Meter scope source
/cfg/mtr/scopetap	S		IN, POST, FILT, PREEQ, POSTEQ, PREFDR, GATEK, DYNK, DYNXO, PRETAP, SOLO, MON.A, MON.B	Meter scope source tap point
/cfg/mtr/mtrsfc	N			Meters fader node [Setup→Surface]
/ cfg/mtr/mtrsfc/in	S		PRE, POST	Meters fader section channel tap
/ cfg/mtr/mtrsfc/bus	S		PRE, POST	Meters fader section bus tap
/ cfg/mtr/mtrsfc/main	S		PRE, POST	Meters fader section main tap
/ cfg/mtr/mtrsfc/mtx	S		PRE, POST	Meters fader section matrix tap
/ cfg/mtr/mtrsfc/dca	S		PRE, POST	Meters fader section DCA tap
/cfg/mtr/mtrpage	N			Meters page node (Meters screen)
/ cfg/mtr/mtrpage/in	S		PRE, POST	Meters page channels tap
/ cfg/mtr/mtrpage/bus	S		PRE, POST	Meters page bus tap
/ cfg/mtr/mtrpage/main	S		PRE, POST	Meters page mains tap
/ cfg/mtr/mtrpage/mtx	S		PRE, POST	Meters page matrix tap
/ cfg/mtr/mtrpage/dca	S		PRE, POST	Meters page DCA tap
/ cfg/mtr/mainmtr	S		MAIN.1.. MAIN.4, MTX.1.. MTX.8, SEL_CH	Main meter
/ cfg/mtr/mainpos	S		AUTO, PRE, POST	Main position
/cfg/talk	N			Talkback config node
/cfg/talk/assign	S		OFF, CH40, AUX8	Talkback assignments
/cfg/talk/$lvl	F	-144..10	-oo..10 in 1024 steps	Talkback level (dB) [RO]
/cfg/talk/indiv	I	0..1		Use individual Bus/Main TB send levels
/cfg/talk/A	N			Talkback A node
/cfg/talk/A/$on	I	0..1		Talkback A off/on
/cfg/talk/A/mode	S		AUTO, PUSH, LATCH	Talkback A mode
/cfg/talk/A/mondim	I	0..1		Talkback A monitor dim
/cfg/talk/A/busdim	F	0..40	41 steps	Talkback A bus dim
/cfg/talk/A/B1	I	0..1		Talkback A bus 1 assign
/cfg/talk/A/B2	I	0..1		Talkback A bus 2 assign
/cfg/talk/A/B3	I	0..1		Talkback A bus 3 assign
/cfg/talk/A/B4	I	0..1		Talkback A bus 4 assign
/cfg/talk/A/B5	I	0..1		Talkback A bus 5 assign
/cfg/talk/A/B6	I	0..1		Talkback A bus 6 assign
/cfg/talk/A/B7	I	0..1		Talkback A bus 7 assign
/cfg/talk/A/B8	I	0..1		Talkback A bus 8 assign
/cfg/talk/A/B9	I	0..1		Talkback A bus 9 assign
/cfg/talk/A/B10	I	0..1		Talkback A bus 10 assign
/cfg/talk/A/B11	I	0..1		Talkback A bus 11 assign
/cfg/talk/A/B12	I	0..1		Talkback A bus 12 assign
/cfg/talk/A/B13	I	0..1		Talkback A bus 13 assign
/cfg/talk/A/B14	I	0..1		Talkback A bus 14 assign
/cfg/talk/A/B15	I	0..1		Talkback A bus 15 assign
/cfg/talk/A/B16	I	0..1		Talkback A bus 16 assign

/cfg/talk/A/M1	I	0..1		Talkback A main 1 assign
/cfg/talk/A/M2	I	0..1		Talkback A main 2 assign
/cfg/talk/A/M3	I	0..1		Talkback A main 3 assign
/cfg/talk/A/M4	I	0..1		Talkback A main 4 assign
/cfg/talk/B	N			Talkback B node
/cfg/talk/B/$on	I	0..1		Talkback B off/on
/cfg/talk/B/mode	S		AUTO, PUSH, LATCH	Talkback B mode
/cfg/talk/B/mondim	I	0..1		Talkback B monitor dim
/cfg/talk/B/busdim	F	0..40	41 steps	Talkback B bus dim
/cfg/talk/B/B1	I	0..1		Talkback B bus 1 assign
/cfg/talk/B/B2	I	0..1		Talkback B bus 2 assign
/cfg/talk/B/B3	I	0..1		Talkback B bus 3 assign
/cfg/talk/B/B4	I	0..1		Talkback B bus 4 assign
/cfg/talk/B/B5	I	0..1		Talkback B bus 5 assign
/cfg/talk/B/B6	I	0..1		Talkback B bus 6 assign
/cfg/talk/B/B7	I	0..1		Talkback B bus 7 assign
/cfg/talk/B/B8	I	0..1		Talkback B bus 8 assign
/cfg/talk/B/B9	I	0..1		Talkback B bus 9 assign
/cfg/talk/B/B10	I	0..1		Talkback B bus 10 assign
/cfg/talk/B/B11	I	0..1		Talkback B bus 11 assign
/cfg/talk/B/B12	I	0..1		Talkback B bus 12 assign
/cfg/talk/B/B13	I	0..1		Talkback B bus 13 assign
/cfg/talk/B/B14	I	0..1		Talkback B bus 14 assign
/cfg/talk/B/B15	I	0..1		Talkback B bus 15 assign
/cfg/talk/B/B16	I	0..1		Talkback B bus 16 assign
/cfg/talk/B/M1	I	0..1		Talkback B main 1 assign
/cfg/talk/B/M2	I	0..1		Talkback B main 2 assign
/cfg/talk/B/M3	I	0..1		Talkback B main 3 assign
/cfg/talk/B/M4	I	0..1		Talkback B main 4 assign
/cfg/amix	N			Automixing node
/cfg/amix/x	I	0..1		Automix X group enable
/cfg/amix/y	I	0..1		Automix Y group enable

System Settings

Command	Type	Range	Text	Description
/$syscfg	N			System configuration node
/$syscfg/consolename	S		16 chars max	Console name
/$syscfg/logflags	S		256 char max	Log flags
/$syscfg/ipmode	S		DHCP, STATIC	IP Mode
/$syscfg/ip0	I	0..255		IP first number
/$syscfg/ip1	I	0..255		IP second number
/$syscfg/ip2	I	0..255		IP third number
/$syscfg/ip3	I	0..255		IP fourth number
/$syscfg/msk0	I	0..255		IP mask first number
/$syscfg/msk1	I	0..255		IP mask second number
/$syscfg/msk2	I	0..255		IP mask third number
/$syscfg/msk3	I	0..255		IP mask fourth numbe
/$syscfg/gw0	I	0..255		IP gateway first number
/$syscfg/gw1	I	0..255		IP gateway second number
/$syscfg/gw2	I	0..255		IP gateway third number
/$syscfg/gw3	I	0..255		IP gateway fourth number
/$syscfg/$ipapply	I	0..1		IP applied
/$syscfg/$firmware	S		64 chars max	Firmware version number [RO]
/$syscfg/$serial	S		32 chars max	Serial number [RO]
/$syscfg/$cnscfg	S		64 chars max	Console configuration/build type string [RO], typically start with “wing”, “wing-rack”, “wing-compact”
/$syscfg/$cnsmdl	S		32 chars max	Console Model right to Console Name in Setup screen [RO], typically “ngc-full” for the full sized desk, can also be “wing-bk”, ”wing-rack”, “wing-compact”
/$syscfg/$chwversion	S		32 chars max	Main board HW version [RO]
/$syscfg /tcplock	I	0..1		Prevent modifications from TCP input
/$syscfg/usbh_spd	S		FS, HS	USB driver speed setting Full Speed, High Speed[13]
/$syscfg/$usbspd_act	S		FS, HS	USB driver speed setting Full Speed, High Speed [RO]
/$syscfg/timefmt	S		24H, 12H	Time format
/$syscfg/datefmt	S		YMD, DMY	Date format
/$syscfg/opt_mod	S		NONE, DANTE, WSG	Installed optional module [RO]

[13] When in FS, record is limited to 2 tracks/16bits. 4 tracks/24bits playing at once from USB stick may be affected; USB 3.1 capable memory sticks are recommended.

Input/Output Settings

Command	Type	Range	Text	Description
/io	N			Input/Output node
/io/altsw	I	0..1		Main/Alt switch
/io/autoaltovr	I	0..1		Global Input Select Override
/io/in	N			Input node
/io/in/LCL	N			Local Input node
/io/in/LCL/1	N	1..24[14]		Local Input 1 node
/io/in/LCL/1/mode	S		M, ST, M/S	Local Input 1 mode
/io/in/LCL/1/g	F	-3..45.5	98 steps	Local Input 1 gain (dB)
/io/in/LCL/1/vph	I	0..1		Local Input 1 phantom
/io/in/LCL/1/mute	I	0..1		Local Input 1 mute
/io/in/LCL/1/pol	I	0..1		Local Input 1 polarity
/io/in/LCL/1/col	I	1..12		Local Input 1 color
/io/in/LCL/1/name	S		16 chars max	Local Input 1 name
/io/in/LCL/1/icon	I	0..999		Local Input 1 icon (indexed)
/io/in/LCL/1/tags	S		80 chars max	Local Input 1 tags
/io/in/LCL/1/$ha	I	0..5		Local input 1 ha type [RO]
/io/in/LCL/1/rmt	S		OFF, AES A, AES B, AES C	Local input 1 remote control
/io/in/LCL/1/$ract	I	0..1		Local input 1 remote active [RO]
/io/in/LCL/1/$rdest	S		7 chars max	Local input 1 remote dest [RO]
/io/in/LCL/1/rcvc	I	0..1		Local input 1 remote customizations sync
/io/in/LCL/1/$mute	I	0..2		Local input 1 mute [RO]
/io/in/AUX	N			Aux Input node
/io/in/AUX/1	N	1..8		Aux Input 1 node
/io/in/AUX/1/mode	S		M, ST, M/S	Aux Input 1 mode
/io/in/AUX/1/mute	I	0..1		Aux Input 1 mute
/io/in/AUX/1/pol	I	0..1		Aux Input 1 polarity
/io/in/AUX/1/col	I	1..12		Aux Input 1 color
/io/in/AUX/1/name	S		16 chars max	Aux Input 1 name
/io/in/AUX/1/icon	I	0..999		Aux Input 1 icon (indexed)
/io/in/AUX/1/tags	S		80 chars max	Aux Input 1 tags
/io/in/AUX/1/$mute	I	0..2		Aux input 1 mute [RO]
/io/in/A	N			AES50 A Input node
/io/in/A/1	N	1..48		AES50 A Input 1 node
/io/in/A/1/mode	S		M, ST, M/S	AES50 A Input 1 mode
/io/in/A/1/g	F	-3..45.5	98 steps	AES50 A Input 1 gain (dB)
/io/in/A/1/vph	I	0..1		AES50 A Input 1 phantom power
/io/in/A/1/mute	I	0..1		AES50 A Input 1 mute
/io/in/A/1/pol	I	0..1		AES50 A Input 1 polarity
/io/in/A/1/col	I	1..12		AES50 A Input 1 color
/io/in/A/1/name	S		16 chars max	AES50 A Input 1 name
/io/in/A/1/icon	I	0..999		AES50 A Input 1 icon (indexed)
/io/in/A/1/tags	S		80 chars max	AES50 A Input 1 tags
/io/in/A/1/$ha	I	0..5		AES50 A input 1 ha type [RO]
/io/in/A/1/rmt	S		OFF, AES A, AES B, AES C	AES50 A input 1 remote control
/io/in/A/1/$ract	I	0..1		AES50 A input 1 remote active [RO]
/io/in/A/1/$rdest	S		7 chars max	AES50 A input 1 remote dest [RO]

[14] All 24 local inputs may not be available depending on the console model

/io/in/A/1/rcvc	I	0..1		AES50 A input 1 remote customizations sync
/io/in/A/1/$mute	I	0..2		AES50 A input 1 mute [RO]
/io/in/B	N			AES50 B Input node
/io/in/B/1	N	1..48		AES50 B Input 1 node
/io/in/B/1/mode	S		M, ST, M/S	AES50 B Input 1 mode
/io/in/B/1/g	F	-3..45.5	98 steps	AES50 B Input 1 gain (dB)
/io/in/B/1/vph	I	0..1		AES50 B Input 1 phantom power
/io/in/B/1/mute	I	0..1		AES50 B Input 1 mute
/io/in/B/1/pol	I	0..1		AES50 B Input 1 polarity
/io/in/B/1/col	I	1..12		AES50 B Input 1 color
/io/in/B/1/name	S		16 chars max	AES50 B Input 1 name
/io/in/B/1/icon	I	0..999		AES50 B Input 1 icon (indexed)
/io/in/B/1/tags	S		80 chars max	AES50 B Input 1 tags
/io/in/B/1/$ha	I	0..5		AES50 B input 1 ha type [RO]
/io/in/B/1/rmt	S		OFF, AES A, AES B, AES C	AES50 B input 1 remote control
/io/in/B/1/$ract	I	0..1		AES50 B input 1 remote active [RO]
/io/in/B/1/$rdest	S		7 chars max	AES50 B input 1 remote dest [RO]
/io/in/B/1/rcvc	I	0..1		AES50 B input 1 remote customizations sync
/io/in/B/1/$mute	I	0..2		AES50 B input 1 mute [RO]
/io/in/C	N			AES50 C Input node
/io/in/C/1	N	1..48		AES50 C Input 1 node
/io/in/C/1/mode	S		M, ST, M/S	AES50 C Input 1 mode
/io/in/C/1/g	F	-3..45.5	98 steps	AES50 C Input 1 gain (dB)
/io/in/C/1/vph	I	0..1		AES50 C Input 1 phantom power
/io/in/C/1/mute	I	0..1		AES50 C Input 1 mute
/io/in/C/1/pol	I	0..1		AES50 C Input 1 polarity
/io/in/C/1/col	I	1..12		AES50 C Input 1 color
/io/in/C/1/name	S		16 chars max	AES50 C Input 1 name
/io/in/C/1/icon	I	0..999		AES50 C Input 1 icon (indexed)
/io/in/C/1/tags	S		80 chars max	AES50 C Input 1 tags
/io/in/C/1/$ha	I	0..4		AES50 C input 1 ha type [RO]
/io/in/C/1/rmt	S		OFF, AES A, AES B, AES C	AES50 C input 1 remote control
/io/in/C/1/$ract	I	0..1		AES50 C input 1 remote active [RO]
/io/in/C/1/$rdest	S		7 chars max	AES50 C input 1 remote dest [RO]
/io/in/C/1/rcvc	I	0..1		AES50 C input 1 remote customizations sync
/io/in/C/1/$mute	I	0..2		AES50 C input 1 mute [RO]
/io/in/SC	N			StageConnect Input node
/io/in/SC/1	N	1..32		StageConnect Input 1 node
/io/in/SC/1/mode	S		M, ST, M/S	StageConnect Input 1 mode
/io/in/SC/1/mute	I	0..1		StageConnect Input 1 mute
/io/in/SC/1/pol	I	0..1		StageConnect Input 1 polarity
/io/in/SC/1/col	I	1..12		StageConnect Input 1 color
/io/in/SC/1/name	S		16 chars max	StageConnect Input 1 name
/io/in/SC/1/icon	I	0..999		StageConnect Input 1 icon (indexed)
/io/in/SC/1/tags	S		80 chars max	StageConnect Input 1 tags
/io/in/SC/1/$mute	I	0..2		StageConnect 1 mute [RO]
/io/in/USB	N			USB Input node
/io/in/USB/1	N	1..48		USB Input 1 node

/io/in/USB/1/mode	S		M, ST, M/S	USB Input 1 mode
/io/in/USB/1/mute	I	0..1		USB Input 1 mute
/io/in/USB/1/pol	I	0..1		USB Input 1 polarity
/io/in/USB/1/col	I	1..12		USB Input 1 color
/io/in/USB/1/name	S		16 chars max	USB Input 1 name
/io/in/USB/1/icon	I	0..999		USB Input 1 icon (indexed)
/io/in/USB/1/tags	S		80 chars max	USB Input 1 tags
/io/in/USB/1/$mute	I	0..2		USB Input 1 mute [RO]
/io/in/CRD	N			Card Input node
/io/in/CRD/1	N	1..64		Card Input 1 node
/io/in/CRD/1/mode	S		M, ST, M/S	Card Input 1 mode
/io/in/CRD/1/mute	I	0..1		Card Input 1 mute
/io/in/CRD/1/pol	I	0..1		Card Input 1 polarity
/io/in/CRD/1/col	I	1..12		Card Input 1 color
/io/in/CRD/1/name	S		16 chars max	Card Input 1 name
/io/in/CRD/1/icon	I	0..999		Card Input 1 icon (indexed)
/io/in/CRD/1/tags	S		80 chars max	Card Input 1 tags
/io/in/CRD/1/$mute	I	0..2		Card Input 1 mute [RO]
/io/in/MOD	N			Module Input node
/io/in/MOD/1	N	1..64		Module Input 1 node
/io/in/MOD/1/mode	S		M, ST, M/S	Module Input 1 node
/io/in/MOD/1/mute	I	0..1		Module Input 1 mute
/io/in/MOD/1/pol	I	0..1		Module Input 1 polarity
/io/in/MOD/1/col	I	1..12		Module Input 1 color
/io/in/MOD/1/name	S		16 chars max	Module Input 1 name
/io/in/MOD/1/icon	I	0..999		Module Input 1 icon (indexed)
/io/in/MOD/1/tags	S		80 chars max	Module Input 1 tags
/io/in/MOD/1/$mute	I	0..2		Module Input 1 mute [RO]
/io/in/PLAY	N			USB Player Input node
/io/in/PLAY/1	N	1..4		USB Player Input 1 node
/io/in/PLAY/1/mode	S		M, ST, M/S	USB Player Input 1 mode
/io/in/PLAY/1/mute	I	0..1		USB Player Input 1 mute
/io/in/PLAY/1/pol	I	0..1		USB Player Input 1 polarity
/io/in/PLAY/1/col	I	1..12		USB Player Input 1 color
/io/in/PLAY/1/name	S		16 chars max	USB Player Input 1 name
/io/in/PLAY/1/icon	I	0..999		USB Player Input 1 icon (indexed)
/io/in/PLAY/1/tags	S		80 chars max	USB Player Input 1 tags
/io/in/PLAY/1/$mute	I	0..2		USB Player Input 1 mute [RO]
/io/in/AES	N			AES/EBU Input node
/io/in/AES/1	N	1..2		AES/EBU Input 1 node
/io/in/AES/1/mode	S		M, ST, M/S	AES/EBU Input 1 mode
/io/in/AES/1/mute	I	0..1		AES/EBU Input 1 mute
/io/in/AES/1/pol	I	0..1		AES/EBU Input 1 polarity
/io/in/AES/1/col	I	1..12		AES/EBU Input 1 color
/io/in/AES/1/name	S		16 chars max	AES/EBU Input 1 name
/io/in/AES/1/icon	I	0..999		AES/EBU Input 1 icon (indexed)
/io/in/AES/1/tags	S		80 chars max	AES/EBU Input 1 tags
/io/in/AES/1/$mute	I	0..2		AES/EBU Input 1 mute [RO]
/io/in/USR	N			User Signal Input node

/io/in/USR/1	N	1..48		User Signal Input 1 node 1-24 are User Signals, 25-48 are User Patches
/io/in/USR/1/mode	S		M, ST, M/S	User Signal Input 1 mode
/io/in/USR/1/mute	I	0..1		User Signal Input 1 mute
/io/in/USR/1/pol	I	0..1		User Signal Input 1 polarity
/io/in/USR/1/col	I	1..12		User Signal Input 1 color
/io/in/USR/1/name	S		16 chars max	User Signal Input 1 name
/io/in/USR/1/icon	I	0..999		User Signal Input 1 icon (indexed)
/io/in/USR/1/tags	S		80 chars max	User Signal Input 1 tags
/io/in/USR/1/$mute	I	0..2		User Signal Input 1 mute [RO]
/io/in/USR/1/user	N			User Signal 1 source node
/io/in/USR/1/user/grp	S		OFF, CH, AUX, BUS, MAIN, MTX	User Signal 1 source group
/io/in/USR/1/user/in	I	1..40		User Signal 1 source number
/io/in/USR/1/user/tap[15]	S		PRE, POST	User Signal 1 source tap point
/io/in/USR/1/user/lr[16]	S		L+R, L, R	User Signal 1 source take
/io/in/OSC	N			Oscillator Input node
/io/in/OSC/1	N	1..2		Oscillator Input 1 node
/io/in/OSC/1/mode	S		M, ST, M/S	Oscillator Input 1 mode [RO]
/io/in/OSC/1/mute	I	0..1		Oscillator Input 1 mute
/io/in/OSC/1/col	I	1..12		Oscillator Input 1 color
/io/in/OSC/1/name	S		16 chars max	Oscillator Input 1 name
/io/in/OSC/1/icon	I	0..999		Oscillator Input 1 icon (indexed)
/io/in/OSC/1/tags	S		80 chars max	Oscillator Input 1 tags
/io/in/OSC/1/$mute	I	0..2		Oscillator Input 1 mute [RO]
/io/in/OSC/1/osc	N			Oscillator 1 source node
/io/in/OSC/1/osc/lvl	F	-40..6	69 steps	Oscillator 1 source level
/io/in/OSC/1/osc/mode	S		SINE, PINK, WHITE	Oscillator 1 source mode
/io/in/OSC/1/osc/f	F	20…20000	2323 steps	Oscillator 1 source frequency
/io/in/$BUS	N			Bus Input node
/io/in/$BUS/1	N	1..32		Bus Input 1 node
/io/in/$BUS/1/mode	S		M, ST, M/S	Bus Input 1 mode [RO]
/io/in/$BUS/1/col	I	1..12		Bus Input 1 color [RO]
/io/in/$BUS/1/name	S		16 chars max	Bus Input 1 name [RO]
/io/in/$BUS/1/icon	I	0..999		Bus Input 1 icon [RO]
/io/in/$BUS/1/tags	S		80 chars max	Bus Input 1 tags [RO]
/io/in/$MAIN	N			Main Input node
/io/in/$MAIN/1	N	1..8		Main Input 1 node
/io/in/$MAIN/1/mode	S		M, ST, M/S	Main Input 1 mode [RO]
/io/in/$MAIN/1/col	I	1..12		Main Input 1 color [RO]
/io/in/$MAIN/1/name	S		16 chars max	Main Input 1 name [RO]
/io/in/$MAIN/1/icon	I	0..999		Main Input 1 icon [RO]
/io/in/$MAIN/1/tags	S		80 chars max	Main Input 1 tags [RO]
/io/in/$MTX	N			Matrix Input node
/io/in/$MTX/1	N	1..16		Matrix Input 1 node
/io/in/$MTX/1/mode	S		M, ST, M/S	Matrix Input 1 mode [RO]

[15] Only for nodes 1..24 [User Signals]
[16] Only for nodes 1..24 [User Signals]

/io/in/$MTX/1/col	I	1..12		Matrix Input 1 color [RO]
/io/in/$MTX/1/name	S		16 chars max	Matrix Input 1 name [RO]
/io/in/$MTX/1/icon	I	0..999		Matrix Input 1 icon [RO]
/io/in/$MTX/1/tags	S		80 chars max	Matrix Input 1 tags [RO]
/io/in/$SEND	N			FX Send Input node
/io/in/$SEND/1	N	1..32		FX Send Input 1 node
/io/in/$SEND/1/mode	S		M, ST, M/S	FX Send Input 1 mode [RO]
/io/in/$SEND/1/col	I	1..12		FX Send Input 1 color [RO]
/io/in/$SEND/1/name	S		16 chars max	FX Send Input 1 name [RO]
/io/in/$SEND/1/icon	I	0..999		FX Send Input 1 icon [RO]
/io/in/$SEND/1/tags	S		80 chars max	FX Send Input 1 tags [RO]
/io/in/$MON	N			Monitor Input node
/io/in/$MON/1	N	1..4		Monitor Input 1 node
/io/in/$MON/1/mode	S		M, ST, M/S	Monitor Input 1 mode [RO]
/io/in/$MON/1/col	I	1..12		Monitor Input 1 color [RO]
/io/in/$MON/1/name	S		16 chars max	Monitor Input 1 name [RO]
/io/in/$MON/1/icon	I	0..999		Monitor Input 1 icon [RO]
/io/in/$MON/1/tags	S		80 chars max	Monitor Input 1 tags [RO]
/io/out	N			Output node
/io/out/LCL	N			Local Output node
/io/out/LCL/1	N	1..8		Local Output 1 node
/io/out/LCL/1/grp	S		OFF, LCL, AUX, A, B, C, SC, USB, CRD, MOD, PLAY, AES, USR, OSC, BUS, MAIN, MTX, SEND, MON	Local Output 1 group
/io/out/LCL/1/in	I	1..64		Local Output 1 input
/io/out/AUX	N			Aux Output node
/io/out/AUX/1	N	1..8		Aux Output 1 node
/io/out/AUX/1/grp	S		OFF, LCL, AUX, A, B, C, SC, USB, CRD, MOD, PLAY, AES, USR, OSC, BUS, MAIN, MTX, SEND, MON	Aux Output 1 group
/io/out/AUX/1/in	I	1..64		Aux Output 1 input
/io/out/A	N			AES50 A Output node
/io/out/A/1	N	1..48		AES50 A Output 1 node
/io/out/A/1/grp	S		OFF, LCL, AUX, A, B, C, SC, USB, CRD, MOD, PLAY, AES, USR, OSC, BUS, MAIN, MTX, SEND, MON	AES50 A Output 1 group
/io/out/A/1/in	I	1..64		AES50 A Output 1 input
/io/out/B	N			AES50 B Output node
/io/out/B/1	N	1..48		AES50 B Output 1 node
/io/out/B/1/grp	S		OFF, LCL, AUX, A, B, C, SC, USB, CRD, MOD, PLAY, AES, USR, OSC, BUS, MAIN, MTX, SEND, MON	AES50 B Output 1 group
/io/out/B/1/in	I	1..64		AES50 B Output 1 input
/io/out/C	N			AES50 C Output node
/io/out/C/1	N	1..48		AES50 C Output 1 node
/io/out/C/1/grp	S		OFF, LCL, AUX, A, B, C, SC, USB, CRD, MOD, PLAY, AES, USR, OSC, BUS, MAIN, MTX, SEND, MON	AES50 C Output 1 group

/io/out/C/1/in	I	1..64		AES50 C Output 1 input
/io/out/SC	N			StageConnect Output node
/io/out/SC/1	N	1..32		StageConnect Output 1 node
/io/out/SC/1/grp	S		OFF, LCL, AUX, A, B, C, SC, USB, CRD, MOD, PLAY, AES, USR, OSC, BUS, MAIN, MTX, SEND, MON	StageConnect Output 1 group
/io/out/SC/1/in	I	1..64		StageConnect Output 1 input
/io/out/USB	N			USB Output Audio node
/io/out/USB/1	N	1..48		USB Output Audio 1 node
/io/out/USB/1/grp	S		OFF, LCL, AUX, A, B, C, SC, USB, CRD, MOD, PLAY, AES, USR, OSC, BUS, MAIN, MTX, SEND, MON	USB Output Audio 1 group
/io/out/USB/1/in	I	1..64		USB Output Audio 1 input
/io/out/CRD	N			Card Output node
/io/out/CRD/1	N	1..64		Card Output 1 node
/io/out/CRD/1/grp	S		OFF, LCL, AUX, A, B, C, SC, USB, CRD, MOD, PLAY, AES, USR, OSC, BUS, MAIN, MTX, SEND, MON	Card Output 1 group
/io/out/CRD/1/in	I	1..64		Card Output 1 input
/io/out/MOD	N			Module Output node
/io/out/MOD/1	N	1..64		Module Output 1 node
/io/out/MOD/1/grp	S		OFF, LCL, AUX, A, B, C, SC, USB, CRD, MOD, PLAY, AES, USR, OSC, BUS, MAIN, MTX, SEND, MON	Module Output 1 group
/io/out/MOD/1/in	I	1..64		Module Output 1 input
/io/out/REC	N			USB Record Output node
/io/out/REC/1	N	1..4		USB Record Output 1 node
/io/out/REC/1/grp	S		OFF, LCL, AUX, A, B, C, SC, USB, CRD, MOD, PLAY, AES, USR, OSC, BUS, MAIN, MTX, SEND, MON	USB Record Output 1 group
/io/out/REC/1/in	I	1..64		USB Record Output 1 input
/io/out/AES	N			AES/EBU Output node
/io/out/AES/1	N	1..2		AES/EBU Output 1 node
/io/out/AES/1/grp	S		OFF, LCL, AUX, A, B, C, SC, USB, CRD, MOD, PLAY, AES, USR, OSC, BUS, MAIN, MTX, SEND, MON	AES/EBU Output 1 group
/io/out/AES/1/in	I	1..64		AES/EBU Output 1 input

Channel Settings

Command	Type	Range	Text	Description
/ch	N			Channel node
/ch/1	N	1..40		Channel 1 node
/ch/1/in	N			Channel 1 input node
/ch/1/in/set	N			Channel 1 input set node
/ch/1/in/set/$mode	S		M, ST, M/S	Channel 1 input mode [RO]
/ch/1/in/set/srcauto	I	0..1		Channel 1 input auto source switch
/ch/1/in/set/altsrc	I	0..1		Channel 1 input main/alt switch
/ch/1/in/set/inv	I	0..1		Channel 1 input phase invert switch
/ch/1/in/set/trim	F	-18..18	361 steps	Channel 1 input trim (dB)
/ch/1/in/set/bal	F	-9..9	181 steps	Channel 1 input balance (dB)
/ch/1/in/set/$g	F	-2.5..45 -3.0..45.5	20 steps (LCL) 98 steps (AES)	Channel 1 input gain (dB) – depends on source type
/ch/1/in/set/$vph	I	0..1		Channel 1 input phantom power – depends on source type
/ch/1/in/set/dlymode	S		M, MS, SMP	Channel 1 input delay mode (meters, ms, samples)
/ch/1/in/set/dly	F	0..150	1501 steps	Channel 1 input delay (meters)
/ch/1/in/set/dlyon	I	0..1		Channel 1 input delay
/ch/1/in/conn	N			Channel 1 input connection node
/ch/1/in/conn/grp	S		OFF, LCL, AUX, A, B, C, SC, USB, CRD, MOD, PLAY, AES, USR, OSC, BUS, MAIN, MTX	Channel 1 main input connection group
/ch/1/in/conn/in	I	1..64		Channel 1 main input connection group index
/ch/1/in/conn/altgrp	S		OFF, LCL, AUX, A, B, C, SC, USB, CRD, MOD, PLAY, AES, USR, OSC, BUS, MAIN, MTX	Channel 1 alt input connection group
/ch/1/in/conn/altin	I	1..64		Channel 1 alt input connection group index
/ch/1/flt	N			Channel 1 filter node
/ch/1/flt/lc	I	0..1		Channel 1 low cut switch
/ch/1/flt/lcf	F	20..2000	641 steps	Channel 1 low cut frequency (Hz)
/ch/1/flt/hc	I	0..1		Channel 1 high cut switch
/ch/1/flt/hcf	F	50..20000	833 steps	Channel 1 high cut frequency (Hz)
/ch/1/flt/tf	I	0..1		Channel 1 tool filter switch
/ch/1/flt/mdl	S		TILT, MAX, AP1, AP2	Channel 1 filter model (see Appendix on Filter plugins for parameters details, OSC patterns in italic below correspond to TILT)
/ch/1/flt/tilt	F	-6..6	49 steps	Channel 1 tilt level (dB)
/ch/1/clink	I	0..1		Channel 1 custom link
/ch/1/col	I	1..12		Channel 1 color
/ch/1/name	S		16 chars max	Channel 1 name
/ch/1/icon	I	0..999		Channel 1 icon
/ch/1/led	I	0..1		Channel 1 scribble light
/ch/1/$col	I	1..12		Channel 1 color [RO] reflects linked source or current strip value
/ch/1/$name	S		16 chars max	Channel 1 name [RO] reflects linked source or current strip value

/ch/1/$icon	I	0..999		Channel 1 icon [RO] reflects linked source or current strip value
/ch/1/mute	I	0..1		Channel 1 mute
/ch/1/fdr	F	-144..10	-oo..10 in 1024 steps	Channel 1 fader
/ch/1/pan	F	-100..100	201 steps	Channel 1 pan
/ch/1/wid	F	-150..150	61 steps	Channel 1 width (%)
/ch/1/$solo	I	0..1		Channel 1 solo switch
/ch/1/$sololed	I	0..2		Channel 1 solo LED [RO]
/ch/1/solosafe	I	0..1		Channel 1 solo safe
/ch/1/mon	S		A, B, A+B	Channel 1 monitor mode
/ch/1/proc	S		GEDI, GEID, GIED, IGED, GDEI, GDIE, GIDE, IGDE, EGDI, EGID, EIGD, IEGD, EDGI, EDIG, EIDG, IEDG, DEGI, DEIG, DIEG, IDEG, DGEI, DGIE, DIGE, IDGE	Channel 1 process order (G: gate, E: EQ, D: Dynamics, I: Insert
/ch/1/ptap	S		IN, FILT, 3, 4, 5, PFL, AFL, POST	Channel 1 pretap (to sends)
/ch/1/$presolo	I	0..1		Channel 1 presolo
/ch/1/peq	N			Channel 1 PreSend EQ node
/ch/1/peq/on	I	0..1		Channel 1 PEQ switch
/ch/1/peq/1g	F	-15..15	301 steps	Channel 1 PEQ band 1 gain (dB)
/ch/1/peq/1f	F	20..20000	960 steps	Channel 1 PEQ band 1 frequency (Hz)
/ch/1/peq/1q	F	0.44..10	181 steps	Channel 1 PEQ band 1 Q
/ch/1/peq/2g	F	-15..15	301 steps	Channel 1 PEQ band 2 gain 9dB)
/ch/1/peq/2f	F	20..20000	960 steps	Channel 1 PEQ band 2 frequency (Hz)
/ch/1/peq/2q	F	0.44..10	181 steps	Channel 1 PEQ band 2 Q
/ch/1/peq/3g	F	-15..15	301 steps	Channel 1 PEQ band 3 gain 9dB)
/ch/1/peq/3f	F	20..20000	960 steps	Channel 1 PEQ band 3 frequency (Hz)
/ch/1/peq/3q	F	0.44..10	181 steps	Channel 1 PEQ band 3 Q
/ch/1/gate	N			Channel 1 gate node
/ch/1/gate/on	I	0..1		Channel 1 gate switch
/ch/1/gate/mdl	S		GATE, DUCK, E88, 9000G, D241, DS902, WAVE, DEQ, WARM, 76LA, LA, RIDE, PSE	Channel 1 gate model (see Appendix on Gate plugins for parameters details, OSC patterns in italic below correspond to GATE)
/ch/1/gate/thr	F	-80..0	161 steps	Channel 1 gate threshold (dB)
/ch/1/gate/range	F	3..60	115 steps	Channel 1 gate range (dB)
/ch/1/gate/att	F	0..120	121 steps	Channel 1 gate attack (ms)
/ch/1/gate/hld	F	0..200	200 steps	Channel 1 gate hold (ms)
/ch/1/gate/rel	F	4..4000	130 steps	Channel 1 gate release(ms)
/ch/1/gate/acc	F	0..100	21 steps	Channel 1 gate accent (5)
/ch/1/gate/ratio	S		1:1.5, 1:2, 1:3, 1:4, GATE	Channel 1 gate ratio
/ch/1/gatesc	N			Channel 1 gate sidechain node
/ch/1/gatesc/type	S		Off, LP12, HP12, BP	Channel 1 gate sidechain type
/ch/1/gatesc/f	F	20..20000	961 steps	Channel 1 gate sidechain frequency (Hz)
/ch/1/gatesc/q	F	0.44..10	181 steps	Channel 1 gate sidechain Q
/ch/1/gatesc/src	S		SHELF, Ch.1..Ch.40	Channel 1 gate sidechain source
/ch/1/gatesc/tap	S		IN, FILT, 3, 4, 5, PFL, AFL, POST	Channel 1 gate sidechain tap
/ch/1/gatesc/$solo	I	0..1		Channel 1 gate sidechain solo
/ch/1/eq	N			Channel 1 EQ node

/ch/1/eq/on	I	0..1		Channel 1 EQ switch
/ch/1/eq/mdl	S		STD, SOUL, E88, E84, F110, PULSAR, MACH4	Channel 1 EQ model (see Appendix on EQ plugins for parameters details, OSC patterns in italic below correspond to STD)
/ch/1/eq/mix	F	0..125	126 steps	Channel 1 EQ mix (%)
/ch/1/eq/$solo	I	0..1		Channel 1 EQ solo
/ch/1/eq/$solobd	I	0..6		Channel 1 EQ solo band
/ch/1/eq/lg	F	-15..15	301 steps	Channel 1 EQ low gain (dB)
/ch/1/eq/lf	F	20..2000	641 steps	Channel 1 EQ low frequency (Hz)
/ch/1/eq/lq	F	0.44..10	181 steps	Channel 1 EQ low Q
/ch/1/eq/leq	S		PEQ, SHV	Channel 1 EQ low type
/ch/1/eq/1g	F	-15..15	301 steps	Channel 1 EQ band 1 gain (dB)
/ch/1/eq/1f	F	20..20000	961 steps	Channel 1 EQ band 1 frequency (Hz)
/ch/1/eq/1q	F	0.44..10	181 steps	Channel 1 EQ band 1 Q
/ch/1/eq/2g	F	-15..15	301 steps	Channel 1 EQ band 2 gain (dB)
/ch/1/eq/2f	F	20..20000	961 steps	Channel 1 EQ band 2 frequency (Hz)
/ch/1/eq/2q	F	0.44..10	181 steps	Channel 1 EQ band 2 Q
/ch/1/eq/3g	F	-15..15	301 steps	Channel 1 EQ band 3 gain (dB)
/ch/1/eq/3f	F	20..20000	961 steps	Channel 1 EQ band 3 frequency (Hz)
/ch/1/eq/3q	F	0.44..10	181 steps	Channel 1 EQ band 3 Q
/ch/1/eq/4g	F	-15..15	301 steps	Channel 1 EQ band 4 gain (dB)
/ch/1/eq/4f	F	20..20000	961 steps	Channel 1 EQ band 4 frequency (Hz)
/ch/1/eq/4q	F	0.44..10	181 steps	Channel 1 EQ band 4 Q
/ch/1/eq/hg	F	-15..15	301 steps	Channel 1 EQ high gain (dB)
/ch/1/eq/hf	F	50..20000	833 steps	Channel 1 EQ high frequency (Hz)
/ch/1/eq/hq	F	0.44..10	181 steps	Channel 1 EQ high Q
/ch/1/eq/heq	S		SHV, PEQ	Channel 1 EQ high type
/ch/1/dyn	N			Channel 1 dynamic (compressor) node
/ch/1/dyn/on	I	0..1		Channel 1 compressor switch
/ch/1/dyn/mdl	S		COMP, EXP, B160, B560, D241, ECL33, 9000C, SBUS, RED3, 76LA, LA, F670, BLISS, NSTR, WAVE, RIDE, 2250, L100	Channel 1 compressor model (see Appendix on Compressor plugins for parameters details, OSC patterns in italic below correspond to COMP)
/ch/1/dyn/mix	F	0..100	101 steps	Channel 1 compressor mix (%)
/ch/1/dyn/gain	F	-6..12	37 steps	Channel 1 compressor gain (dB)
/ch/1/dyn/thr	F	-60..0	121 steps	Channel 1 compressor threshold (dB)
/ch/1/dyn/ratio	S		1.1, 1.2, 1.3, 1.5, 1.7, 2.0, 2.5, 3.0, 3.5, 4.0, 5.0, 6.0, 8.0, 10, 20, 50, 100	Channel 1 compressor ratio
/ch/1/dyn/knee	I	0..5		Channel 1 compressor knee
/ch/1/dyn/det	S		PEAK, RMS	Channel 1 compressor detect
/ch/1/dyn/att	F	0..120	121 steps	Channel 1 compressor attack (ms)
/ch/1/dyn/hld	F	1..200	200 steps	Channel 1 compressor hold (ms)
/ch/1/dyn/rel	F	4..4000	130 steps	Channel 1 compressor release (ms)
/ch/1/dyn/env	S		LIN, LOG	Channel 1 compressor envelope
/ch/1/dyn/auto	I	0..1		Channel 1 compressor auto switch
/ch/1/dynxo	N			Channel 1 compressor crossover node
/ch/1/dynxo/depth	F	0..20	41 steps	Channel 1 compressor crossover depth (dB)
/ch/1/dynxo/type	S		OFF, LO6, LO12, HI6, HI12, PC	Channel 1 compressor crossover type

/ch/1/dynxo/f	F	20..20000	901 steps	Channel 1 compressor crossover frequency (Hz)
/ch/1/dynxo/$solo	I	0..1		Channel 1 compressor crossover solo
/ch/1/dynsc	N			Channel 1 compressor sidechain node
/ch/1/dynsc/type	S		Off, LP12, HP12, BP	Channel 1 compressor sidechain type
/ch/1/dynsc/f	F	20..20000	901 steps	Channel 1 compressor sidechain frequency (Hz)
/ch/1/dynsc/q	F	0.44..10	181 steps	Channel 1 compressor sidechain Q
/ch/1/dynsc/src	S		SELF, CH.1..CH.40	Channel 1 compressor sidechain source
/ch/1/dynsc/tap	S		IN, FILT, 3, 4, 5, PFL, AFL, POST	Channel 1 compressor sidechain tap
/ch/1/dynsc/$solo	I	0..1		Channel 1 compressor sidechain solo
/ch/1/preins	N			Channel 1 pre-insert node
/ch/1/preins/on	I	0..1		Channel 1 pre-insert switch
/ch/1/preins/ins	S		NONE, FX1..FX16	Channel 1 pre-insert FX slot
/ch/1/preins/$stat	S		-, OK, N/A	Channel 1 pre-insert status [RO]
/ch/1/main	N			Channel 1 Main node
/ch/1/main/1	N	1..4		Channel 1 Main 1 node
/ch/1/main/1/on	I	0..1		Channel 1 Main 1 on switch
/ch/1/main/1/lvl	F	-144..10	-oo..10 in 1024 steps	Channel 1 Main 1 fader level (dB)
/ch/1/main/pre	I	0..1		Channel 1 sent pre fader to Main 1
/ch/1/send	N			Channel 1 sends node
/ch/1/send/1	N	1..16		Channel 1 sends 1 node
/ch/1/send/1/on	I	0..1		Channel 1 sends 1 on switch
/ch/1/send/1/lvl	F	-144..10	-oo..10 in 1024 steps	Channel 1 sends 1 fader level (dB)
/ch/1/send/1/pon	I	0..1		Channel 1 sends 1 pre always on switch
/ch/1/send/1/ind	I	0..1		Channel 1 sends 1 individual tap (0=link to bus)
/ch/1/send/1/mode	S		PRE, POST, GRP	Channel 1 sends 1 mode
/ch/1/send/1/plink	I	0..1		Channel 1 sends 1 pan link (0=individual)
/ch/1/send/1/pan	F	-100..100	201 steps	Channel 1 sends 1 pan
/ch/1/send/1/wid	F	-150..150	61 steps	Channel 1 sends 1 width (%)
/ch/1/postins	N			Channel 1 post insert node
/ch/1/postins/on	I	0..1		Channel 1 post insert on switch
/ch/1/postins/mode	S		FX, AUTO_X, AUTO_Y	Channel 1 post insert mode
/ch/1/postins/ins	S		NONE, FX1..FX16	Channel 1 post insert FX slot
/ch/1/postins/w	F	-12..12	241 steps	Channel 1 post insert autogain weight
/ch/1/postins/$stat	S		-, OK, N/A	Channel 1 post insert status [RO]
/ch/1/tags	S		80 chars max	Channel 1 tags[17]
/ch/1/$fdr	F	-144..10	-oo..10 in 1024 steps	Channel 1 fader level as affected by dca (dB) [RO]
/ch/1/$mute	I	0..2		Channel 1 mute [RO]
/ch/1/$muteovr	I	0..1		Channel 1 mute override

[17] Tags #D1..#D16 are 'reserved' for DCA1..16 assignment

Aux Settings

Command	Type	Range	Text	Description
/aux	N			Aux node
/aux/1	N	1..8		Aux 1 node
/aux/1/in	N			Aux 1 input node
/aux/1/in/set	N			Aux 1 input set node
/aux/1/in/set/$mode	S		M, ST, M/S	Aux 1 input mode [RO]
/aux/1/in/set/srcauto	I	0..1		Aux 1 input auto source switch
/aux/1/in/set/altsrc	I	0..1		Aux 1 input main/alt switch
/aux/1/in/set/inv	I	0..1		Aux 1 input phase invert switch
/aux/1/in/set/trim	F	-18..18	361 steps	Aux 1 input trim (dB)
/aux/1/in/set/bal	F	-9..9	181 steps	Aux 1 input balance (dB)
/aux/1/in/set/$g	F	-3..45	98 steps	Aux 1 input gain (dB)
/aux/1/in/set/$vph	I	0..1		Aux 1 input phantom power
/aux/1/in/conn	N			Aux 1 input connection node
/aux/1/in/conn/grp	S		OFF, LCL, AUX, A, B, C, SC, USB, CRD, MOD, PLAY, AES, USR, OSC, BUS, MAIN, MTX	Aux 1 input connection group
/aux/1/in/conn/in	I	1..64		Aux 1 input connection group index
/aux/1/in/conn/altgrp	S		OFF, LCL, AUX, A, B, C, SC, USB, CRD, MOD, PLAY, AES, USR, OSC, BUS, MAIN, MTX	Aux 1 alt input connection group
/aux/1/in/conn/altin	I	1..64		Aux 1 alt input connection group index
/aux/1/clink	I	0..1		Aux 1 custom link
/aux/1/col	I	1..12		Aux 1 color
/aux/1/name	S		16 chars max	Aux 1 name
/aux/1/icon	I	0..999		Aux 1 icon
/aux/1/led	I	0..1		Aux 1 scribble light
/aux/1/$col	I	1..12		Aux 1 color [RO] reflects linked source or current strip value
/aux/1/$name	S		16 chars max	Aux 1 name [RO] reflects linked source or current strip value
/aux/1/$icon	I	0..999		Aux 1 icon [RO] reflects linked source or current strip value
/aux/1/mute	I	0..1		Aux 1 mute
/aux/1/fdr	F	-144..10	-oo..10 in 1024 steps	Aux 1 fader level (dB)
/aux/1/pan	F	-100..100	201 steps	Aux 1 pan
/aux/1/wid	F	-150..150	61 steps	Aux 1 width (%)
/aux/1/$solo	I	0..1		Aux 1 solo
/aux/1/$sololed	I	0..2		Aux 1 solo LED [RO]
/aux/1/solosafe	I	0..1		Aux 1 solo safe
/aux/1/mon	S		A, B, A+B	Aux 1 monitor mode
/aux/1/eq	N			Aux 1 EQ node
/aux/1/eq/on	I	0..1		Aux 1 EQ switch
/aux/1/eq/mdl	S		STD, SOUL, E88, E84, F110, PULSAR	Aux 1 EQ model (see Appendix on EQ plugins for parameters details, OSC patterns in italic below correspond to STD)
/ aux /1/eq/mix	F	0..125	126 steps	Aux 1 EQ mix (%)
/ aux /1/eq/$solo	I	0..1		Aux 1 EQ solo

/ aux /1/eq/$solobd	I	0..6		Aux 1 EQ solo band
/ aux */1/eq/lg*	F	-15..15	301 steps	Aux 1 EQ low gain (dB)
/ aux */1/eq/lf*	F	20..2000	641 steps	Aux 1 EQ low frequency (Hz)
/ aux */1/eq/lq*	F	0.44..10	181 steps	Aux 1 EQ low Q
/ aux */1/eq/leq*	S		SHV, PEQ, CUT	Aux 1 EQ low type
/ aux */1/eq/1g*	F	-15..15	301 steps	Aux 1 EQ band 1 gain (dB)
/ aux */1/eq/1f*	F	20..20000	961 steps	Aux 1 EQ band 1 frequency (Hz)
/ aux */1/eq/1q*	F	0.44..10	181 steps	Aux 1 EQ band 1 Q
/ aux */1/eq/2g*	F	-15..15	301 steps	Aux 1 EQ band 2 gain (dB)
/ aux */1/eq/2f*	F	20..20000	961 steps	Aux 1 EQ band 2 frequency (Hz)
/ aux */1/eq/2q*	F	0.44..10	181 steps	Aux 1 EQ band 2 Q
/ aux */1/eq/3g*	F	-15..15	301 steps	Aux 1 EQ band 3 gain (dB)
/ aux */1/eq/3f*	F	20..20000	961 steps	Aux 1 EQ band 3 frequency (Hz)
/ aux */1/eq/3q*	F	0.44..10	181 steps	Aux 1 EQ band 3 Q
/ aux */1/eq/4g*	F	-15..15	301 steps	Aux 1 EQ band 4 gain (dB)
/ aux */1/eq/4f*	F	20..20000	961 steps	Aux 1 EQ band 4 frequency (Hz)
/ aux */1/eq/4q*	F	0.44..10	181 steps	Aux 1 EQ band 4 Q
/ aux */1/eq/hg*	F	-15..15	301 steps	Aux 1 EQ high gain (dB)
/ aux */1/eq/hf*	F	50..20000	833 steps	Aux 1 EQ high frequency (Hz)
/ aux */1/eq/hq*	F	0.44..10	181 steps	Aux 1 EQ high Q
/ aux */1/eq/heq*	S		SHV, PEQ, CUT	Aux 1 EQ high type
/aux/1/dyn	N			Aux 1 dynamic (compressor) node
/aux/1/dyn/on	I	0..1		Aux 1 compressor switch
/aux/1/dyn/thr	F	-36..12	97 steps	Aux 1 compressor threshold (dB)
/aux/1/dyn/depth	F	0..20	41 steps	Aux 1 compressor depth (dB)
/aux/1/dyn/fast	I	0..1		Aux 1 compressor fast switch
/aux/1/dyn/peak	I	0..1		Aux 1 compressor peak switch
/aux/1/dyn/ingain	F	0..100	101 steps	Aux 1 compressor input gain
/aux/1/dyn/cpeak	F	0..100	101 steps	Aux 1 compressor peak
/aux/1/dyn/cmode	S		COMP, LIM	Aux 1 compressor mode
/aux/1/preins	N			Aux 1 pre-insert node
/aux/1/preins/on	I	0..1		Aux 1 pre-insert switch
/aux/1/preins/ins	S		NONE, FX1..FX16	Aux 1 pre-insert FX slot
/aux/1/preins/$stat	S		-, OK, N/A	Aux 1 pre-insert status [RO]
/aux/1/main	N			Aux 1 Main node
/aux/1/main/1	N	1..4		Aux 1 Main 1 node
/aux/1/main/1/on	I	0..1		Aux 1 Main 1 on switch
/aux/1/main/1/lvl	F	-144..10	-oo..10 in 1024 steps	Aux 1 Main 1 fader level (dB)
/aux/1/main/1/pre	I	0..1		Aux 1 sent pre fader to Main 1
/aux/1/send	N			Aux 1 sends node
/aux/1/send/1	N	1..16		Aux 1 sends 1 node
/aux/1/send/1/on	I	0..1		Aux 1 sends 1 on switch
/aux/1/send/1/lvl	F	-144..10	-oo..10 in 1024 steps	Aux 1 sends 1 fader level (dB)
/aux/1/send/1/pon	I	0..1		Aux 1 sends 1 pre always on switch
/aux/1/send/1/ind	I	0..1		Aux 1 sends 1 individual link (0=link to bus)
/aux/1/send/1/mode	S		PRE, POST, GRP	Aux 1 sends 1 mode
/aux/1/send/1/plink	I	0..1		Aux 1 sends 1 pan link (0=individual)
/aux/1/send/1/pan	F	-100..100	201 steps	Aux 1 sends 1 pan
/aux/1/send/1/wid	F	-150..150	61 steps	Aux 1 sends 1 width (%)

/aux/1/tags	S	80 chars max		Aux 1 tags[18]
/aux/1/$fdr	F	-144..10	-oo..10 in 1024 steps	Aux 1 fader level as affected by dca (dB)[RO]
/aux/1/$mute	I	0..2		Aux 1 mute {RO]
/aux/1/$muteovr	I	0..1		Aux 1 mute override

[18] Tags #D1..#D16 are 'reserved' for DCA1..16 assignment

Bus Settings

Command	Type	Range	Text	Description
/bus	N			Bus node
/bus/1	N	1..16		Bus 1 node
/bus/1/in	N			Bus 1 input node
/bus/1/in/set	N			Bus 1 input set node
/bus/1/in/set/inv	I	0..1		Bus 1 input phase invert
/bus/1/in/set/trim	F	-18..18	361 steps	Bus 1 input trim (dB)
/bus/1/in/set/bal	F	-9..9	181 steps	Bus 1 input balance (dB)
/bus/1/col	I	1..12		Bus 1 color
/bus/1/name	S		16 chars max	Bus 1 name
/bus/1/icon	I	0..999		Bus 1 icon
/bus/1/led	I	0..1		Bus 1 scribble light
/bus/1/$col	I	1..12		Bus 1 color [RO] reflects linked source or current strip value
/bus/1/$name	S		16 chars max	Bus 1 name [RO] reflects linked source or current strip value
/bus/1/$icon	I	0..999		Bus 1 icon [RO] reflects linked source or current strip value
/bus/1/busmono	I	0..1		Bus 1 mono switch
/bus/1/mute	I	0..1		Bus 1 mute
/bus/1/fdr	F	-144..10	-oo..10 in 1024 steps	Bus 1 fader level (dB)
/bus/1/pan	F	-100..100	201 steps	Bus 1 pan
/bus/1/wid	F	-150..150	61 steps	Bus 1 width (%)
/bus/1/$solo	I	0..1		Bus 1 solo
/bus/1/$sololed	I	0..2		Bus 1 solo LED {RO]
/bus/1/mon	S		A, B, A+B	Bus 1 monitor mode
/bus/1/busmode	S		PRE, POST, GRP	Bus 1 mode
/bus/1/eq	N			Bus 1 EQ node
/bus/1/eq/on	I	0..1		Bus 1 EQ on switch
/bus/1/eq/mdl	S		STD, SOUL, E88, E84, F110, PULSAR, PIA	Bus 1 EQ model (see Appendix on EQ plugins for parameters details, OSC patterns in italic below correspond to STD)
/bus/1/eq/mix	F	0..100	126 steps	Bus 1 EQ mix
/bus/1/eq/$solo	I	0..1		Bus 1 EQ solo
/bus/1/eq/$solobd	I	0..1		Bus 1 EQ band solo
/bus/1/eq/lg	F	-15..15	301 steps	Bus 1 EQ low gain (dB)
/bus/1/eq/lf	F	20..2000	641 steps	Bus 1 EQ low frequency (Hz)
/bus/1/eq/lq	F	0.44..10	181 steps	Bus 1 EQ low Q
/bus/1/eq/leq	S		PEQ, SHV, CUT, BW6, BW12, BS12, LR12, BW18, BW24, BS24, LR24, BW48, LR48	Bus 1 EQ low type
/bus/1/eq/1g	F	-15..15	301 steps	Bus 1 EQ band 1 gain (dB)
/bus/1/eq/1f	F	20..20000	961 steps	Bus 1 EQ band 1 frequency (Hz)
/bus/1/eq/1q	F	0.44..10	181 steps	Bus 1 EQ band 1 Q
/bus/1/eq/2g	F	-15..15	301 steps	Bus 1 EQ band 2 gain (dB)
/bus/1/eq/2f	F	20..20000	961 steps	Bus 1 EQ band 2 frequency (Hz)
/bus/1/eq/2q	F	0.44..10	181 steps	Bus 1 EQ band 2 Q
/bus/1/eq/3g	F	-15..15	301 steps	Bus 1 EQ band 3 gain (dB)
/bus/1/eq/3f	F	20..20000	961 steps	Bus 1 EQ band 3 frequency (Hz)
/bus/1/eq/3q	F	0.44..10	181 steps	Bus 1 EQ band 3 Q
/bus/1/eq/4g	F	-15..15	301 steps	Bus 1 EQ band 4 gain (dB)

/bus/1/eq/4f	F	20..20000	961 steps	Bus 1 EQ band 4 frequency (Hz)
/bus/1/eq/4q	F	0.44..10	181 steps	Bus 1 EQ band 4 Q
/bus/1/eq/5g	F	-15..15	301 steps	Bus 1 EQ band 5 gain (dB)
/bus/1/eq/5f	F	20..20000	961 steps	Bus 1 EQ band 5 frequency (Hz)
/bus/1/eq/5q	F	0.44..10	181 steps	Bus 1 EQ band 5 Q
/bus/1/eq/6g	F	-15..15	301 steps	Bus 1 EQ band 6 gain (dB)
/bus/1/eq/6f	F	20..20000	961 steps	Bus 1 EQ band 6 frequency (Hz)
/bus/1/eq/6q	F	0.44..10	181 steps	Bus 1 EQ band 6 Q
/bus/1/eq/hg	F	-15..15	301 steps	Bus 1 EQ high gain (dB)
/bus/1/eq/hf	F	50..20000	833 steps	Bus 1 EQ high frequency (Hz)
/bus/1/eq/hq	F	0.44..10	181 steps	Bus 1 EQ high Q
/bus/1/eq/heq	S		PEQ, SHV, CUT, BW6, BW12, BS12, LR12, BW18, BW24, BS24, LR24, BW48, LR48	Bus 1 EQ high type
/bus/1/eq/tilt	F	-6..6	49 steps	Bus 1 EQ tilt level
/bus/1/dyn	N			Bus 1 dynamic (compressor) node
/bus/1/dyn/on	I	0..1		Bus 1 compressor switch
/bus/1/dyn/mdl	S		COMP, EXP, B160, B560, D241, ECL33, 9000C, SBUS, RED3, 76LA, LA, F670, BLISS, NSTR, WAVE, RIDE, 2250, L100	Bus 1 compressor model, (see Appendix on Compressor plugins for parameters details, OSC patterns in italic below correspond to COMP)
/bus/1/dyn/mix	F	0..100	101 steps	Bus 1 compressor mix (%)
/bus/1/dyn/gain	F	-6..12	37 steps	Bus 1 compressor gain (dB)
/bus/1/dyn/thr	F	-60..0	121 steps	Bus 1 compressor threshold (dB)
/bus/1/dyn/ratio	F	1.1..100		Bus 1 compressor ratio
/bus/1/dyn/knee	I	0..5		Bus 1 compressor knee
/bus/1/dyn/det	S		PEAK, RMS	Bus 1 compressor detect
/bus/1/dyn/att	F	0..120	121 steps	Bus 1 compressor attack (ms)
/bus/1/dyn/hld	F	1..200	200 steps	Bus 1 compressor hold (ms)
/bus/1/dyn/rel	F	4..4000	130 steps	Bus 1 compressor release (ms)
/bus/1/dyn/env	S		Lin, Log	Bus 1 compressor envelope
/bus/1/dyn/auto	I	0..1		Bus 1 compressor auto switch
/bus/1/dynxo	N			Bus 1 compressor crossover node
/bus/1/dynxo/depth	F	0..20	41 steps	Bus 1 compressor crossover depth (dB)
/bus/1/dynxo/type	S		OFF, LO6, LO12, HI6, HI12, PC	Bus 1 compressor crossover type
/bus/1/dynxo/f	F	20..20000	901 steps	Bus 1 compressor crossover frequency (Hz)
/bus/1/dynxo/$solo	I	0..1		Bus 1 compressor crossover solo
/bus/1/dynsc	N			Bus 1 compressor sidechain node
/bus/1/dynsc/type	S		OFF, LP12, HP12, BP	Bus 1 compressor sidechain type
/bus/1/dynsc/f	F	20..20000	901 steps	Bus 1 compressor sidechain frequency (Hz)
/bus/1/dynsc/q	F	0.44..10	181 steps	Bus 1 compressor sidechain Q
/bus/1/dynsc/src	S		SELF, BUS.1..BUS.16, MAIN.1..MAIN.4, MTX.1..MTX.8, AUX.1..AUX.8	Bus 1 compressor sidechain source
/bus/1/dynsc/tap	S		BUS, DYN, PFL, AFL, EQ, INS2	Bus 1 compressor sidechain tap
/bus/1/dynsc/$solo	I	0..1		Bus 1 compressor sidechain solo
/bus/1/preins	N			Bus 1 pre-insert node
/bus/1/preins/on	I	0..1		Bus 1 pre-insert switch

/bus/1/preins/ins	S		NONE, FX1..FX16	Bus 1 pre-insert FX slot
/bus/1/preins/$stat	S		-, OK, N/A	Bus 1 pre-insert status [RO]
/bus/1/main	N			Bus 1 Main node
/bus/1/main/1	N	1..4		Bus 1 Main 1 node
/bus/1/main/1/on	I	0..1		Bus 1 Main 1 on switch
/bus/1/main/1/lvl	F	-144..10	-oo..10 in 1024 steps	Bus 1 Main 1 fader level (dB)
/bus/1/main/1/pre	I	0..1		Bus 1 sent pre fader to Main 1
/bus/1/send	N			Bus 1 sends node
/bus/1/send/1	N	1..8		Bus 1 sends 1 node
/bus/1/send/1/on	I	0..1		Bus 1 sends 1 on switch
/bus/1/send/1/lvl	F	-144..10	-oo..10 in 1024 steps	Bus 1 sends 1 fader level (dB)
/bus/1/send/1/pre	I	0..1		Bus 1 sends 1 pre/post switch
/bus/1/send/MX1	N			Bus 1 matrix 1 sends node
/bus/1/send/MX1/on	I	0..1		Bus 1 matrix 1 on switch
/bus/1/send/MX1/lvl	F	-144..10	-oo..10 in 1024 steps	Bus 1 matrix 1 fader level (dB)
/bus/1/send/MX1/pre	I	0..1		Bus 1 matrix 1 pre/post switch
/bus/1/send/MX2	N			Bus 1 matrix 2 sends node
/bus/1/send/MX2/on	I	0..1		Bus 1 matrix 2 on switch
/bus/1/send/MX2/lvl	F	-144..10	-oo..10 in 1024 steps	Bus 1 matrix 2 fader level (dB)
/bus/1/send/MX2/pre	I	0..1		Bus 1 matrix 2 pre/post switch
/bus/1/send/MX3	N			Bus 1 matrix 3 sends node
/bus/1/send/MX3/on	I	0..1		Bus 1 matrix 3 on switch
/bus/1/send/MX3/lvl	F	-144..10	-oo..10 in 1024 steps	Bus 1 matrix 3 fader level (dB)
/bus/1/send/MX3/pre	I	0..1		Bus 1 matrix 3 pre/post switch
/bus/1/send/MX4	N			Bus 1 matrix 4 sends node
/bus/1/send/MX4/on	I	0..1		Bus 1 matrix 4 on switch
/bus/1/send/MX4/lvl	F	-144..10	-oo..10 in 1024 steps	Bus 1 matrix 4 fader level (dB)
/bus/1/send/MX4/pre	I	0..1		Bus 1 matrix 4 pre/post switch
/bus/1/send/MX5	N			Bus 1 matrix 5 sends node
/bus/1/send/MX5/on	I	0..1		Bus 1 matrix 5 on switch
/bus/1/send/MX5/lvl	F	-144..10	-oo..10 in 1024 steps	Bus 1 matrix 5 fader level (dB)
/bus/1/send/MX5/pre	I	0..1		Bus 1 matrix 5 pre/post switch
/bus/1/send/MX6	N			Bus 1 matrix 6 sends node
/bus/1/send/MX6/on	I	0..1		Bus 1 matrix 6 on switch
/bus/1/send/MX6/lvl	F	-144..10	-oo..10 in 1024 steps	Bus 1 matrix 6 fader level (dB)
/bus/1/send/MX6/pre	I	0..1		Bus 1 matrix 6 pre/post switch
/bus/1/send/MX7	N			Bus 1 matrix 7 sends node
/bus/1/send/MX7/on	I	0..1		Bus 1 matrix 7 on switch
/bus/1/send/MX7/lvl	F	-144..10	-oo..10 in 1024 steps	Bus 1 matrix 7 fader level (dB)
/bus/1/send/MX7/pre	I	0..1		Bus 1 matrix 7 pre/post switch
/bus/1/send/MX8	N			Bus 1 matrix 8 sends node
/bus/1/send/MX8/on	I	0..1		Bus 1 matrix 8 on switch
/bus/1/send/MX8/lvl	F	-144..10	-oo..10 in 1024 steps	Bus 1 matrix 8 fader level (dB)
/bus/1/send/MX8/pre	I	0..1		Bus 1 matrix 8 pre/post switch

/bus/1/postins	N			Bus 1 post insert node
/bus/1/postins/on	I	0..1		Bus 1 post insert on switch
/bus/1/postins/ins	S		NONE, FX1..FX16	Bus 1 post insert mode
/bus/1/postins/$stat	S		-, OK, N/A	Bus 1 post insert status [RO]
/bus/1/tags	S		80 chars max	Bus 1 tags[19]
/bus/1/$fdr	F	-144..10	-oo..10 in 1024 steps	Bus 1 fader level as affected by dca (dB)[RO]
/bus/1/$mute	I	0..2		Bus 1 mute [RO]
/bus/1/$muteovr	I	0..1		Bus 1 mute override

[19] Tags #D1..#D16 are 'reserved' for DCA1..16 assignment

Mains Settings

Command	Type	Range	Text	Description
/main	N			Main node
/main/1	N	1..4		Main 1 node
/main/1/in	N			Main 1 input node
/main/1/in/set	N			Main 1 input set node
/main/1/in/set/inv	I	0..1		Main 1 input phase invert switch
/main/1/in/set/trim	F	-18..18	361 steps	Main 1 input trim
/main/1/in/set/bal	F	-9..9	181 steps	Main 1 input balance
/main/1/col	I	1..12		Main 1 color
/main/1/name	S		16 chars max	Main 1 name
/main/1/icon	I	0..999		Main 1 icon
/main/1/led	I	0..1		Main 1 scribble light
/main/1/$col	I	1..12		Main 1 color [RO] reflects linked source or current strip value
/main/1/$name	S		16 chars max	Main 1 name [RO] reflects linked source or current strip value
/main/1/$icon	I	0..999		Main 1 icon [RO] reflects linked source or current strip value
/main/1/busmono	I	0..1		Main 1 mono switch
/main/1/mute	I	0..1		Main 1 mute
/main/1/fdr	F	-144..10	-oo..10 in 1024 steps	Main 1 fader level (dB)
/main/1/pan	F	-100..100	201 steps	Main 1 pan
/main/1/wid	F	-150..150	61 steps	Main 1 width (%)
/main/1/$solo	I	0..1		Main 1 solo switch
/main/1/$sololed	I	0..2		Main 1 solo LED [RO]
/main/1/mon	S		A, B, A+B	Main 1 monitor mode
/main/1/eq	N			Main 1 EQ node
/main/1/eq/on	I	0..1		Main 1 EQ on switch
/main/1/eq/mdl	S		STD, SOUL, E88, E84, F110, PULSAR, PIA	Main 1 EQ model, (see Appendix on EQ plugins for parameters details, OSC patterns in italic below correspond to STD)
/main/1/eq/mix	F	0..100	126 steps	Main 1 EQ mix
/main/1/eq/$solo	I	0..1		Main 1 EQ solo
/main/1/eq/$solobd	I	0..1		Main 1 EQ band solo
/main/1/eq/lg	F	-15..15	301 steps	Main 1 EQ low gain (dB)
/main/1/eq/lf	F	20..2000	641 steps	Main 1 EQ low frequency (Hz)
/main/1/eq/lq	F	0.44..10	181 steps	Main 1 EQ low Q
/main/1/eq/leq	S		PEQ, SHV, CUT, BW6, BW12, BS12, LR12, BW18, BW24, BS24, LR24, BW48, LR48	Main 1 EQ low type
/main/1/eq/1g	F	-15..15	301 steps	Main 1 EQ band 1 gain (dB)
/main/1/eq/1f	F	20..20000	961 steps	Main 1 EQ band 1 frequency (Hz)
/main/1/eq/1q	F	0.44..10	181 steps	Main 1 EQ band 1 Q
/main/1/eq/2g	F	-15..15	301 steps	Main 1 EQ band 2 gain (dB)
/main/1/eq/2f	F	20..20000	961 steps	Main 1 EQ band 2 frequency (Hz)
/main/1/eq/2q	F	0.44..10	181 steps	Main 1 EQ band 2 Q
/main/1/eq/3g	F	-15..15	301 steps	Main 1 EQ band 3 gain (dB)
/main/1/eq/3f	F	20..20000	961 steps	Main 1 EQ band 3 frequency (Hz)
/main/1/eq/3q	F	0.44..10	181 steps	Main 1 EQ band 3 Q
/main/1/eq/4g	F	-15..15	301 steps	Main 1 EQ band 4 gain (dB)
/main/1/eq/4f	F	20..20000	961 steps	Main 1 EQ band 4 frequency (Hz)

/main/1/eq/4q	F	0.44..10	181 steps	Main 1 EQ band 4 Q
/main/1/eq/5g	F	-15..15	301 steps	Main 1 EQ band 5 gain (dB)
/main/1/eq/5f	F	20..20000	961 steps	Main 1 EQ band 5 frequency (Hz)
/main/1/eq/5q	F	0.44..10	181 steps	Main 1 EQ band 5 Q
/main/1/eq/6g	F	-15..15	301 steps	Main 1 EQ band 6 gain (dB)
/main/1/eq/6f	F	20..20000	961 steps	Main 1 EQ band 6 frequency (Hz)
/main/1/eq/6q	F	0.44..10	181 steps	Main 1 EQ band 6 Q
/main/1/eq/hg	F	-15..15	301 steps	Main 1 EQ high gain (dB)
/main/1/eq/hf	F	50..20000	833 steps	Main 1 EQ high frequency (Hz)
/main/1/eq/hq	F	0.44..10	181 steps	Main 1 EQ high Q
/main/1/eq/heq	S		PEQ, SHV, CUT, BW6, BW12, BS12, LR12, BW18, BW24, BS24, LR24, BW48, LR48	Main 1 EQ high type
/main/1/eq/tilt	F	-6..6	49 steps	Main 1 EQ tilt level
/main/1/dyn	N			Main 1 dynamic (compressor) node
/main/1/dyn/on	I	0..1		Main 1 compressor switch
/main/1/dyn/mdl	S		COMP, EXP, B160, B560, D241, ECL33, 9000C, SBUS, RED3, 76LA, LA, F670, BLISS, NSTR, WAVE, RIDE, 2250, L100	Main 1 compressor switch, (see Appendix on Compressor plugins for parameters details, OSC patterns in italic below correspond to COMP)
/main/1/dyn/mix	F	0..100	101 steps	Main 1 compressor mix (%)
/main/1/dyn/gain	F	-6..12	37 steps	Main 1 compressor gain (dB)
/main/1/dyn/thr	F	-60..0	121 steps	Main 1 compressor threshold (dB)
/main/1/dyn/ratio	F	1.1..100		Main 1 compressor ratio
/main/1/dyn/knee	I	0..5		Main 1 compressor knee
/main/1/dyn/det	S		PEAK, RMS	Main 1 compressor detect
/main/1/dyn/att	F	0..120	121 steps	Main 1 compressor attack (ms)
/main/1/dyn/hld	F	1..200	200 steps	Main 1 compressor hold (ms)
/main/1/dyn/rel	F	4..4000	130 steps	Main 1 compressor release (ms)
/main/1/dyn/env	S		LIN, LOG	Main 1 compressor envelope
/main/1/dyn/auto	I	0..1		Main 1 compressor auto switch
/main/1/dynxo	N			Main 1 compressor crossover node
/main/1/dynxo/depth	F	0..20	41 steps	Main 1 compressor crossover depth (dB)
/main/1/dynxo/type	S		OFF, LO6, LO12, HI6, HI12, PC	Main 1 compressor crossover type
/main/1/dynxo/f	F	20..20000	901 steps	Main 1 compressor crossover frequency (Hz)
/main/1/dynxo/$solo	I	0..1		Main 1 compressor crossover solo
/main/1/dynsc	N			Main 1 compressor sidechain node
/main/1/dynsc/type	S		OFF, LP12, HP12, BP	Main 1 compressor sidechain type
/main/1/dynsc/f	F	20..20000	901 steps	Main 1 compressor sidechain frequency (Hz)
/main/1/dynsc/q	F	0.44..10	181 steps	Main 1 compressor sidechain Q
/main/1/dynsc/src	S		SELF, BUS.1..BUS.16, MAIN.1..MAIN.4, MTX.1..MTX.8, AUX.1..AUX.8	Main 1 compressor sidechain source
/main/1/dynsc/tap	S		BUS, DYN, PFL, AFL, EQ, INS2	Main 1 compressor sidechain tap
/main/1/dynsc/$solo	I	0..1		Main 1 compressor sidechain solo
/main/1/preins	N			Main 1 pre-insert node
/main/1/preins/on	I	0..1		Main 1 pre-insert switch
/main/1/preins/ins	S		NONE, FX1..FX16	Main 1 pre-insert FX slot

/main/1/preins/$stat	S		-, OK, N/A	Main 1 pre-insert status [RO]
/main/1/send	N			Main 1 sends node
/main/1/send/MX1	N			Main 1 matrix sends node
/main/1/send/MX1/on	I	0..1		Main 1 matrix 1 on switch
/main/1/send/MX1/lvl	F	-144..10	-oo..10 in 1024 steps	Main 1 matrix 1 fader level (dB)
/main/1/send/MX1/pre	I	0..1		Main 1 matrix 1 pre/post switch
/main/1/send/MX2	N			Main 2 matrix sends node
/main/1/send/MX2/on	I	0..1		Main 2 matrix 1 on switch
/main/1/send/MX2/lvl	F	-144..10	-oo..10 in 1024 steps	Main 2 matrix 1 fader level (dB)
/main/1/send/MX2/pre	I	0..1		Main 2 matrix 1 pre/post switch
/main/1/send/MX3	N			Main 3 matrix sends node
/main/1/send/MX3/on	I	0..1		Main 3 matrix 1 on switch
/main/1/send/MX3/lvl	F	-144..10	-oo..10 in 1024 steps	Main 3 matrix 1 fader level (dB)
/main/1/send/MX3/pre	I	0..1		Main 3 matrix 1 pre/post switch
/main/1/send/MX4	N			Main 4 matrix sends node
/main/1/send/MX4/on	I	0..1		Main 4 matrix 1 on switch
/main/1/send/MX4/lvl	F	-144..10	-oo..10 in 1024 steps	Main 4 matrix 1 fader level (dB)
/main/1/send/MX4/pre	I	0..1		Main 4 matrix 1 pre/post switch
/main/1/send/MX5	N			Main 5 matrix sends node
/main/1/send/MX5/on	I	0..1		Main 5 matrix 1 on switch
/main/1/send/MX5/lvl	F	-144..10	-oo..10 in 1024 steps	Main 5 matrix 1 fader level (dB)
/main/1/send/MX5/pre	I	0..1		Main 5 matrix 1 pre/post switch
/main/1/send/MX6	N			Main 6 matrix sends node
/main/1/send/MX6/on	I	0..1		Main 6 matrix 1 on switch
/main/1/send/MX6/lvl	F	-144..10	-oo..10 in 1024 steps	Main 6 matrix 1 fader level (dB)
/main/1/send/MX6/pre	I	0..1		Main 6 matrix 1 pre/post switch
/main/1/send/MX7	N			Main 7 matrix sends node
/main/1/send/MX7/on	I	0..1		Main 7 matrix 1 on switch
/main/1/send/MX7/lvl	F	-144..10	-oo..10 in 1024 steps	Main 7 matrix 1 fader level (dB)
/main/1/send/MX7/pre	I	0..1		Main 7 matrix 1 pre/post switch
/main/1/send/MX8	N			Main 8 matrix sends node
/main/1/send/MX8/on	I	0..1		Main 8 matrix 1 on switch
/main/1/send/MX8/lvl	F	-144..10	-oo..10 in 1024 steps	Main 8 matrix 1 fader level (dB)
/main/1/send/MX8/pre	I	0..1		Main 8 matrix 1 pre/post switch
/main/1/postins	N			Main 1 post insert node
/main/1/postins/on	I	0..1		Main 1 post insert on switch
/main/1/postins/ins	S		NONE, FX1..FX16	Main 1 post insert mode
/main/1/postins/$stat	S		-, OK, N/A	Main 1 post insert status [RO]
/main/1/dly	N			Main 1 delay node
/main/1/dly/on	I	0..1		Main 1 delay on switch
/main/1/dly/m	F	0.1..100	1000 steps	Main 1 delay (meters)

/main/1/tags	S		80 chars max	Main 1 tags[20]
/main/1/$fdr	F	-144..10	-oo..10 in 1024 steps	Main 1 fader level as affected by dca (dB)[RO]
/main/1/$mute	I	0..2		Main 1 mute [RO]
/main/1/$muteovr	I	0..1		Main 1 mute override

[20] Tags #D1..#D16 are 'reserved' for DCA1..16 assignment

Matrix Settings

Command	Type	Range	Text	Description
/mtx	N			Matrix node
/mtx/1	N	1..8		Matrix 1 node
/mtx/1/in	N			Matrix 1 input node
/mtx/1/in/set	N			Matrix 1 input set node
/mtx/1/in/set/inv	I	0..1		Matrix 1 input phase invert
/mtx/1/in/set/trim	F	-18..18	361 steps	Matrix 1 input trim
/mtx/1/in/set/bal	F	-9..9	181 steps	Matrix 1 input balance
/mtx/1/dir	N			Matrix 1 direct input signal
/mtx/1/dir/1	N	1..2		Matrix 1 direct in 1 node
/mtx/1/dir/1/on	I	0..1		Matrix 1 direct in 1 on switch
/mtx/1/dir/1/lvl	F	-144..10	-oo..10 in 1024 steps	Matrix 1 direct in 1 fader level (dB)
/mtx/1/dir/1/inv	I	0..1		Matrix 1 direct in 1 invert
/mtx/1/dir/1/in	S		OFF, CH.1..CH.40, AUX.1..AUX.8, AES, MON.A, MON.B, MON.BUS	Matrix 1 direct in 1 input
/mtx/1/dir/1/tap	S		PRE, POST	Matrix 1 direct in 1 tap
/mtx/1/col	I	1..12		Matrix 1 color
/mtx/1/name	S		16 chars max	Matrix 1 name
/mtx/1/icon	I	0..999		Matrix 1 icon
/mtx/1/led	I	0..1		Matrix 1 scribble light
/mtx/1/$col	I	1..12		Matrix 1 color [RO] reflects linked source or current strip value
/mtx/1/$name	S		16 chars max	Matrix 1 name [RO] reflects linked source or current strip value
/mtx/1/$icon	I	0..999		Matrix 1 icon [RO] reflects linked source or current strip value
/mtx/1/busmono	I	0..1		Matrix 1 mono switch
/mtx/1/mute	I	0..1		Matrix 1 mute
/mtx/1/fdr	F	-144..10	-oo..10 in 1024 steps	Matrix 1 fader level (dB)
/mtx/1/pan	F	-100..100	201 steps	Matrix 1 pan
/mtx/1/wid	F	-150..150	61 steps	Matrix 1 width (%)
/mtx/1/$solo	I	0..1		Matrix 1 solo switch
/mtx/1/$sololed	I	0..2		Matrix 1 solo LED [RO]
/mtx/1/mon	S		A, B, A+B	Matrix 1 monitor mode
/mtx/1/eq	N			Matrix 1 EQ node
/mtx/1/eq/on	I	0..1		Matrix 1 EQ on switch
/mtx/1/eq/mdl	S		STD, SOUL, E88, E84, F110, PULSAR, PIA	Matrix 1 EQ model, (see Appendix on EQ plugins for parameters details, OSC patterns in italic below correspond to STD)
/mtx/1/eq/mix	F	0..125	126 steps	Matrix 1 EQ mix (%)
/mtx/1/eq/$solo	I	0..1		Matrix 1 EQ solo
/mtx/1/eq/$solobd	I	0..8		Matrix 1 EQ solo band
/mtx/1/eq/lg	F	-15..15	301 steps	Matrix 1 EQ low gain (dB)
/mtx/1/eq/lf	F	20..2000	641 steps	Matrix 1 EQ low frequency
/mtx/1/eq/lq	F	0.44..10	181 steps	Matrix 1 EQ low Q
/mtx/1/eq/leq	S		SHV, PEQ, CUT	Matrix 1 EQ low type
/mtx/1/eq/1g	F	-15..15	301 steps	Matrix 1 EQ band 1 gain (dB)
/mtx/1/eq/1f	F	20..20000	961 steps	Matrix 1 EQ band 1 frequency (Hz)
/mtx/1/eq/1q	F	0.44..10	181 steps	Matrix 1 EQ band 1 Q
/mtx/1/eq/2g	F	-15..15	301 steps	Matrix 1 EQ band 2 gain (dB)

/mtx/1/eq/2f	F	20..20000	961 steps	Matrix 1 EQ band 2 frequency (Hz)
/mtx/1/eq/2q	F	0.44..10	181 steps	Matrix 1 EQ band 2 Q
/mtx/1/eq/3g	F	-15..15	301 steps	Matrix 1 EQ band 3 gain (dB)
/mtx/1/eq/3f	F	20..20000	961 steps	Matrix 1 EQ band 3 frequency (Hz)
/mtx/1/eq/3q	F	0.44..10	181 steps	Matrix 1 EQ band 3 Q
/mtx/1/eq/4g	F	-15..15	301 steps	Matrix 1 EQ band 4 gain (dB)
/mtx/1/eq/4f	F	20..20000	961 steps	Matrix 1 EQ band 4 frequency (Hz)
/mtx/1/eq/4q	F	0.44..10	181 steps	Matrix 1 EQ band 4 Q
/mtx/1/eq/5g	F	-15..15	301 steps	Matrix 1 EQ band 5 gain (dB)
/mtx/1/eq/5f	F	20..20000	961 steps	Matrix 1 EQ band 5 frequency (Hz)
/mtx/1/eq/5q	F	0.44..10	181 steps	Matrix 1 EQ band 5 Q
/mtx/1/eq/6g	F	-15..15	301 steps	Matrix 1 EQ band 6 gain (dB)
/mtx/1/eq/6f	F	20..20000	961 steps	Matrix 1 EQ band 6 frequency (Hz)
/mtx/1/eq/6q	F	0.44..10	181 steps	Matrix 1 EQ band 6 Q
/mtx/1/eq/hg	F	-15..15	301 steps	Matrix 1 EQ high gain (dB)
/mtx/1/eq/hf	F	50..20000	833 steps	Matrix 1 EQ high frequency (Hz)
/mtx/1/eq/hq	F	0.44..10	181 steps	Matrix 1 EQ high Q
/mtx/1/eq/heq	S		SHV, PEQ, CUT	Matrix 1 EQ high type
/mtx/1/eq/tilt	F	-6..6	49 steps	Matrix 1 EQ tilt level (dB)
/mtx/1/dyn	N			Matrix 1 dynamic (compressor) node
/mtx/1/dyn/on	I	0..1		Matrix 1 compressor switch
/mtx/1/dyn/mdl	S		COMP, EXP, B160, B560, D241, ECL33, 9000C, SBUS, RED3, 76LA, LA, F670, BLISS, NSTR, WAVE, RIDE, 2250, L100	Matrix 1 compressor model, (see Appendix on Compressor plugins for parameters details, OSC patterns in italic below correspond to COMP)
/mtx/1/dyn/mix	F	0..100	101 steps	Matrix 1 compressor mix (%)
/mtx/1/dyn/gain	F	-6..12	37 steps	Matrix 1 compressor gain (dB)
/mtx/1/dyn/thr	F	-60..0	121 steps	Matrix 1 compressor threshold (dB)
/mtx/1/dyn/ratio	F	1.1..100		Matrix 1 compressor ratio
/mtx/1/dyn/knee	I	0..5		Matrix 1 compressor knee
/mtx/1/dyn/det	S		PEAK, RMS	Matrix 1 compressor detect
/mtx/1/dyn/att	F	0..120	121 steps	Matrix 1 compressor attack (ms)
/mtx/1/dyn/hld	F	1..200	200 steps	Matrix 1 compressor hold (ms)
/mtx/1/dyn/rel	F	4..4000	130 steps	Matrix 1 compressor release (ms)
/mtx/1/dyn/env	S		LIN, LOG	Matrix 1 compressor envelope
/mtx/1/dyn/auto	I	0..1		Matrix 1 compressor auto switch
/mtx/1/dynxo	N			Matrix 1 compressor crossover node
/mtx/1/dynxo/depth	F	0..20	41 steps	Matrix 1 compressor crossover depth (dB)
/mtx/1/dynxo/type	S		OFF, LO6, LO12, HI6, HI12, PC	Matrix 1 compressor crossover type
/mtx/1/dynxo/f	F	20..20000	901 steps	Matrix 1 compressor crossover frequency (Hz)
/mtx/1/dynxo/$solo	I	0..1		Matrix 1 compressor crossover solo
/mtx/1/dynsc	N			Matrix 1 compressor sidechain node
/mtx/1/dynsc/type	S		OFF, LP12, HP12, BP	Matrix 1 compressor sidechain type
/mtx/1/dynsc/f	F	20..20000	901 steps	Matrix 1 compressor sidechain frequency (Hz)
/mtx/1/dynsc/q	F	0.44..10	181 steps	Matrix 1 compressor sidechain Q
/mtx/1/dynsc/src	S		SELF, BUS.1..BUS.16, MAIN.1..MAIN.4, MTX.1..MTX.8, AUX.1..AUX.8	Matrix 1 compressor sidechain source
/mtx/1/dynsc/tap	S		BUS, DYN, PFL, AFL, EQ, INS2	Matrix 1 compressor sidechain tap

/mtx/1/dynsc/$solo	I	0..1		Matrix 1 compressor sidechain solo
/mtx/1/preins	N			Matrix 1 pre-insert node
/mtx/1/preins/on	I	0..1		Matrix 1 pre-insert switch
/mtx/1/preins/ins	S		NONE, FX1..FX16	Matrix 1 pre-insert FX slot
/mtx/1/preins/$stat	S		-,OK, N/A	Matrix 1 pre-insert status [RO]
/mtx/1/postins	N			Matrix 1 post insert node
/mtx/1/postins/on	I	0..1		Matrix 1 post insert on switch
/mtx/1/postins/ins	S		NONE, FX1..FX16	Matrix 1 post insert mode
/mtx/1/postins/$stat	S		-,OK, N/A	Matrix 1 post insert status [RO]
/mtx/1/dly	N			Matrix 1 delay node
/mtx/1/dly/on	I	0..1		Matrix 1 delay on switch
/mtx/1/dly/m	F	0.1..100	1000 steps	Maitrix1 delay (meters)
/mtx/1/tags	S		80 chars max	Matrix 1 tags[21]
/mtx/1/$fdr	F	-144..10	-oo..10 in 1024 steps	Matrix 1 fader level as affected by dca (dB)[RO]
/mtx/1/$mute	I	0..2		Matrix 1 mute [RO]
/mtx/1/$muteovr	I	0..1		Matrix 1 mute override

[21] Tags #D1..#D16 are 'reserved' for DCA1..16 assignment

DCA Settings

Command	Type	Range	Text	Description
/dca	N			DCA node
/dca/1	N	1..16		DCA 1 node
/dca/1/name	S		8 chars max	DCA 1 name
/dca/1/col	I	1..12		DCA 1 color
/dca/1/icon	I	0..999		DCA 1 icon
/dca/1/led	I	0..1		DCA 1 scribble light
/dca/1/mute	I	0..1		DCA 1 mute
/dca/1/fdr	F	-144..10	-oo..10 in 1024 steps	DCA 1 fader (dB)
/dca/1/$solo	I	0..1		DCA 1 solo
/dca/1/$sololed	I	0..1		DCA 1 solo LED [RO]
/dca/1/mon	S		A, B, A+B	DCA 1 monitor mode

Mutegroup Settings

Command	Type	Range	Text	Description
/mgrp	N			Mutegroup node
/mgrp/1	N	1..8		Mutegroup 1 node
/mgrp/1/name	S		8 chars max	Mutegroup 1 name
/mgrp/1/mute	I	0..1		Mutegroup 1 mute

Effects Settings

Command	Type	Range	Text	Description
/fx	N			FX node
/fx/1	N	1..16		FX 1 node
/fx/1/mdl	S		For /fx/1../fx/8: NONE, EXT, HALL, ROOM, CHAMBER, PLATE, CONCERT, AMBI, V-ROOM, V-REV, V-PLATE, GATED, REVERSE, DEL/REV, SHIMMER, SPRING, DIMCRS, CHORUS, FLANGER, ST-DL, TAP-DL, TAPE-DL, OILCAN, BBD-DL, PITCH, D-PITCH, VSS3, BPLATE, GEQ, PIA, DOUBLE, PCORR, LIMITER, DE-S2, ENHANCE, EXCITER, P-BASS, ROTARY, PHASER, PANNER, TAPE, MOOD, SUB, RACKAMP, UKROCK, ANGEL, JAZZC, DELUXE, BODY, SOUL, E88, E84, F110, PULSAR, MACH4, C5-CMB, SUB-M, V-IMG, SPKMAN, DEQ3, *EVEN*, *SOUL*, *VINTAGE*, *BUS*, *MASTER* For /fx/9../fx/16: NONE, EXT, GEQ, PIA, DOUBLE, PCORR, LIMITER, DE-S2, ENHANCE, EXCITER, P-BASS, ROTARY, PHASER, PANNER, TAPE, MOOD, SUB, RACKAMP, UKROCK, ANGEL, JAZZC, DELUXE, BODY, SOUL, E88, E84, F110, PULSAR, MACH4, C5-CMB, SUB-M, V-IMG, SPKMAN, DEQ3, *EVEN*, *SOUL*, *VINTAGE*, *BUS*, *MASTER*	FX 1 model (see Appendix for details, graphics and parameter values)
/fx/1/fxmix	F	0..100	101 steps	FX 1 mix % (depends on FX)
/fx/1/$esrc	I	0..400		FX 1 source [RO]
/fx/1/$emode	S		M, ST, M/S	FX 1 mode [RO]
/fx/1/$a_chn	I	0..76		FX 1 channel assigned to it [RO]
/fx/1/$a_pos	I	0..1		FX 1 channel insert (0=pre, 1=post) [RO]
/fx/1/…	/fx/1/… contains up to 64 parameters that depend on the selected model (/fx/1/mdl), as listed in the Appendix section			

Cards Settings

Command	Type	Range	Text	Description
/cards	N			Cards node
/cards/$type	S		NONE, WLIVE, WDANTE, WLINK	Cards type [RO]
/cards/$ver	S		32 chars max	Cards version [RO]
/cards/wlive	N			Cards W-Live node
/cards/wlive/sdlink	S		IND, PAR	Cards W-Live SD parallel mode
/cards/wlive/$actlink	S		IND, PAR	Cards W-Live ACT link [RO]
/cards/wlive/$battstate	S		NONE, GOOD, LOW	Cards W-Live battery status [RO]
/cards/wlive/autoin	S		OFF, 1, 2	Cards W-Live auto input
/cards/wlive/meters	I	0..1		Cards show meters
/cards/wlive/auto_stop	S		KEEP, MAIN, ALT	Cards W-Live settings when Stop
/cards/wlive/auto_play	S		KEEP, MAIN, ALT	Cards W-Live settings when Play
/cards/wlive/auto_rec	S		KEEP, MAIN, ALT	Cards W-Live settings when Rec
/cards/wlive/1	N	1..2		Cards W-Live 1 node
/cards/wlive/1/$ctl	N			Cards W-Live 1 ctl node
/cards/wlive/1/$ctl/control	S		STOP, PPAUSE, PLAY, REC	Cards W-Live 1 control
/cards/wlive/1/$ctl/opensession	I	0..100		Cards W-Live 1 open session #
/cards/wlive/1/$ctl/editmarker	I	0..100		Cards W-Live 1 edit marker (set marker to current PAUSE time or last start PLAY time)
/cards/wlive/1/$ctl/gotomarker	I	0..101		Cards W-Live 1 goto marker # 101 is used to validate stime data
/cards/wlive/1/$ctl/deletemarker	I	0..100		Cards W-Live 1 delete marker #
/cards/wlive/1/$ctl/deletesession	I	0..100		Cards W-Live 1 delete session #
/cards/wlive/1/$ctl/stime	F	0..36000000	36000000 steps	Cards W-Live 1 time (ms). Must be followed by a $ctl/gotomarker 101 to be taken into account
/cards/wlive/1/$ctl/namesession	S		19 chars max	Cards W-Live 1 name session. Works only in STOP mode.
/cards/wlive/1/$ctl/setmarker	I	0..1		Cards W-Live 1 set marker
/cards/wlive/1/$ctl/formatsdcard	I	0..1		Cards W-Live 1 format SD card
/cards/wlive/1/cfg	N			Cards W-Live 1 cfg node
/cards/wlive/1/cfg/rectracks	S		32, 16, 8	Cards W-Live 1 rec tracks
/cards/wlive/1/cfg/playmode	S		PLAY, A->B, LOOP	Cards W-Live 1 play mode
/cards/wlive/1/$stat	N			Cards W-Live 1 status node
/cards/wlive/1/$stat/state	S		STOP, PPAUSE, PLAY, REC	Cards W-Live 1 state [RO]
/cards/wlive/1/$stat/etime	F	0..36000000	36000000 steps	Cards W-Live 1 etime (current time)
/cards/wlive/1/$stat/sdfree	F	0..36000000	36000000 steps	Cards W-Live 1 SD free space
/cards/wlive/1/$stat/sdsize	I	0..1024		Cards W-Live 1 SD size (Gb) [RO]
/cards/wlive/1/$stat/sdstate	S		NONE, READY, PROTECT, ERASE, ERROR	Cards W-Live 1 SD state [RO]
/cards/wlive/1/$stat/sessionlist	S		Ex: 2020-04-04 10:16:36, 2020-01-27 19:59:02, …	Cards W-Live 1 list of session recorded date and time
/cards/wlive/1/$stat/markerlist	S			Cards W-Live 1 current marker time

/cards/wlive/1/$stat/snamelist	S		Ex: CC Hard Candy Fi, CC Mr Jones	Cards W-Live 1 session names list[22] [RO]
/cards/wlive/1/$stat/sessions	I	0..100		Cards W-Live 1 total number of sessions [RO]
/cards/wlive/1/$stat/markers	I	0..100		Cards W-Live 1 total number of markers [RO]
/cards/wlive/1/$stat/sessionlen	F	0..36000000	36000000 steps	Cards W-Live 1 session length [RO]
/cards/wlive/1/$stat/sessionpos	I	0..100		Cards W-Live 1 session position
/cards/wlive/1/$stat/markerpos	I	0..100		Cards W-Live 1 marker position
/cards/wlive/1/$stat/tracks	S		32, 16, 8	Cards W-Live 1 track number in current session [RO]
/cards/wlive/1/$stat/rate	S		44.1, 48	Cards W-Live 1 sample rate [RO]
/cards/wlive/1/$stat/linkedpos	I	0..100		Cards W-Live session link position in the other card (only when a linked session is active)[23]
/cards/wlive/1/$stat/start	F	0..36000000	36000000 steps	Cards W-Live 1 start
/cards/wlive/1/$stat/stop	F	0..36000000	36000000 steps	Cards W-Live 1 stop
/cards/wlive/1/$stat/errormessage	S		32 chars max	Cards W-Live 1 error message [RO]
/cards/wlive/1/$stat/errorcode	I	0..34		Cards W-Live 1 error code [RO]

[22] Only the first name of the list is returned by std OSC command. You must use the node definition command (OSC or native interface) to get the full contents.

[23] There can be a maximum of 100 sessions per card

USB Player Settings

Command	Type	Range	Text	Description
/play[24]	N			USB Player node
/play/$songs	S		List of strings	List of songs [RO][25]
/play/$actlist	S		256 chars max	Path to USB files [RO]
/play/$actidx	I		1..n	Current active entry in the playlist
/play/$actionidx	I		1..n	[RO]
/play/$playfile	S		256 chars max	Full path to a song to play using `/play/$action ,s PLAYFILE`
/play/$action	S		IDLE, STOP, PLAY, PAUSE, NEXT, PREV, PLAYFILE	USB Player action
/play/$actstate	S		STOP, PLAY, PAUSE, ERROR	USB Player active state [RO]
/play/$actfile	S		256 chars max	USB Player active file [RO]
/play/$song	S		64 chars max	USB Player song [RO]
/play/$album	S		64 chars max	USB Player album [RO]
/play/$artist	S		64 chars max	USB Player artist [RO]
/play/$pos	F	0..35999	36000 steps	USB Player position
/play/$total	F	0..35999	36000 steps	USB Player total time [RO]
/play/$resolution	S		16, 24	USB Player resolution [RO]
/play/$channels	S		1, 2, 3, 4	USB Player channels [RO]
/play/$rate	S		44.1, 48	USB Player sample rate [RO]
/play/$format	S		WAV, MP3, FLAC	USB Player format [RO]
/play/repeat	I	0..1		USB Player repeat
/rec	N			USB Recorder node
/rec/$actstate	S		STOP, REC, PAUSE, ERROR	USB Recorder active state
/rec/$actfile	S			USB Recorder active filename
/rec/$action	S		STOP, REC, PAUSE, NEWFILE	USB Recorder action
/rec/path	S			USB Recorder filename path
/rec/resolution	S			USB Recorder resolution
/rec/channels	S			USB Recorder channels
/rec/$time	F			USB Recorder time

[24] These commands are valid only when a playlist is active, and opened.

[25] Will provide only the first element of the list. Use the node level request to get the full list of songs in the playlist, for ex: `/play~~~,s~~?~~~`

Global Settings

/$globals	N			Global Settings node
/$globals/clkrate	F	44100, 48000		Master clock rate
/$globals/clksrc	S		INT, A, B, C, AES, CARD, MOD	Master clock source
/$globals/startmute	I	0..1		Mute outputs on startup
/$globals/usbacfg	S		2/2, 8/8, 16/16, 32/32, 48/48	USB Input/Output configuration
/$globals/sccfg	S		AUTO, 0/32, 1/31, 2/30, 3/29, 4/28, 5/27, 6/26, 7/25, 8/24, 9/23, 10/22, 11/21, 12/20, 13/19, 14/18, 15/17, 16/16, 17/15, 18/14, 19/13, 20/12, 21/11, 22/10, 23/9, 24/8, 25/7, 26/6, 27/5, 28/4, 29/3, 30/2, 31/1, 32/0	SC Configuration
/$globals/harmt	N			HA remote node
/$globals/harmt/a	I	0..1		Enable HA remote on AES-A
/$globals/harmt/b	I	0..1		Enable HA remote on AES-B
/$globals/harmt/c	I	0..1		Enable HA remote on AES-C
/$globals/custsync	N			Custom Sync node
/$globals/custsync /a	I	0..1		Enable Cust Sync on AES-A
/$globals/custsync /b	I	0..1		Enable Cust Sync on AES-B
/$globals/custsync /c	I	0..1		Enable Cust Sync on AES-C

WING ce_data OSC commands list

Control Settings, listed below, form a large set of OSC commands and parameters, all[26] under the ***ce_data*** section in `JSON` snapshot files.

Control Settings

Command	Type	Range	Text	Description
/$ctl	N			Control node
/$ctl/$stat	N			Control status node
/$ctl/$stat/selidx	I	1..76		Channel strip selected ID[27]
/$ctl/$stat/pageidx	I	0..30		Channel page ID
/$ctl/$stat/bandidx	I	1..8		Channel EQ band ID
/$ctl/$stat/sof	I	-1..76		Sends on fader (SoF) status [-1 is the currently selected channel]
/$ctl/$stat/cnslock	S		19 chars when locked 0 chars if unlocked	Console lock [RO] – The console lock string is made of 19 characters `0` or `1` depending on which screen buttons were pressed to lock the console. Characters 1 to 7 map to the buttons on the screen left, starting with `HOME` [`ASSIGN` is char #7], and characters 18 & 19 map to the buttons on the right side of the screen. Other characters are always `0`. Ex: `1001001000000000000` - buttons `HOME`, `ROUTING` and `ASSIGN` have been used to lock the desk.
/$ctl/cfg	N			Control config node
/$ctl/cfg/lights	N			Lights node
/$ctl/cfg/lights/btns	I	0..100		Buttons backlight intensity
/$ctl/cfg/lights/leds	I	5..100		Buttons/LED light intensity
/$ctl/cfg/lights/meters	I	0..100		Meters intensity
/$ctl/cfg/lights/rgbleds	I	0..100		Color LED intensity (scribble lights)
/$ctl/cfg/lights/chlcds	I	5..100		Channel LCD intensity (scribble backlight)
/$ctl/cfg/lights/chlcdctr	I	0..100		Channel LCD contrast (scribble contrast)
/$ctl/cfg/lights/chedit	I	5..100		Channel strip intensity
/$ctl/cfg/lights/main	I	5..100		Touchscreen intensity
/$ctl/cfg/lights/glow	I	0..100		Under console light intensity
/$ctl/cfg/lights/patch	I	0..100		Patch panel light intensity
/$ctl/cfg/lights/lamp	I	0..100		Lamp light intensity
/$ctl/cfg/rta[28]	N			RTA node (view options)
/$ctl/cfg/rta/homedisp	S		OFF, 1/3, FULL	RTA home size/mode
/$ctl/cfg/rta/homecol	S		RD25, RD50, RD75, AM25, AM50, AM75, BL25, BL50, BL75	RTA home color
/$ctl/cfg/rta/hometap	S		IN, EQ, POST	RTA home tap

[26] With the exception of the /$ctl/$stat, and parameters /$ctl/cfg/$noautosave and /$ctl/cfg/savenow

[27] The get command reports values between 0 and 75, but index 1 to 76 should be used when setting values.

[28] See also /cfg/rta commands

/$ctl/cfg/rta/eqdisp	S		Off, 1/4, 1/3, 1/2, OVL/3, OVL	RTA EQ size/mode
/$ctl/cfg/rta/eqcol	S		RD25, RD50, RD75, AM25, AM50, AM75, BL25, BL50, BL75	RTA EQ color
/$ctl/cfg/rta/cheqtap	S		PRE, POST	RTA EQ tap
/$ctl/cfg/rta/chflttap	S		PRE, POST	RTA Channel filter tap
/$globals/muteovr	I	0..1		Chan strip mute overrides mute group
/$ctl/cfg/soloexcl	I	0..1		Solo exclusive
/$ctl/cfg/selfsolo	I	0..1		Select follows solo
/$ctl/cfg/solofsel	I	0..1		Solo follows select
/$ctl/cfg/sof2solo	I	0..1		Bus SOF activates solo
/$ctl/cfg/layerlinkl	I	0..1		User Layer link left/center
/$ctl/cfg/layerlinkr	I	0..1		User Layer link center/right
/$ctl/cfg/autoview	I	0..1		Screen follows channel strip
/$ctl/cfg/csctouch	I	0..1		Channel strip touch select
/$ctl/cfg/autosel_L	I	0..1		Channel auto select left
/$ctl/cfg/autosel_C	I	0..1		Channel auto select center
/$ctl/cfg/autosel_R	I	0..1		Channel auto select right
/$ctl/cfg/fdrbanking	I	0..1		Full fader paging
/$ctl/cfg/soffdr	S		L/C, ALL	SOF Faders (L/C: left/center)
/$ctl/cfg/sofbutton	S		AUTO, ON, FLASH	SOF button mode
/$ctl/cfg/sofframe	I	0..1		SOF frame
/$ctl/cfg/sofmode	I	0..1		Alternative SOF mode
/$ctl/cfg/seldblclick	S		OFF, HOME, BUSFX	Where Double click select takes you
/$ctl/cfg/usrmode	S		BUS, CC	Use F1-F3 as BUS or Custom Control
/$ctl/cfg/mfdr	S		OFF, MAIN.1, .., MAIN.4, MTX.1,.., MTX.8, DCA.1, .., DCA.16	What functionality is assigned to the Compact model fader 13/Main
/$ctl/cfg/cscmode	S		BUS, DCA, MAIN, USER	Operation mode for the 16 buttons of Compact model
/$ctl/cfg/rackmode	S		CH, MGRP, CC, USB, SD-A, SD-B	Operation mode for the 4 channel strip section on Rack model
/$ctl/cfg/busspill	I	0..1		Compact model only: 1: **push**->bus spill, **hold**->bus send 0: **push**->bus send, **hold**->bus spill
/$ctl/cfg/mainspill	I	0..1		Compact model only: 1: **push**->main spill, **hold**->main send 0: **push**->main send, **hold**->main spill
/$ctl/cfg/mtxspill	I	0..1		Compact model only: 1: **push**->mtx spill, **hold**->mtx send 0: **push**->mtx send, **hold**->mtx spill
/$ctl/cfg/dcaspill	I	0..1		1: **push**->dca spill, **hold**->dca [un]select 0: **push**->dca show, **hold**->dca [un]select
/$ctl/cfg/showfdr	I	0..1		Temporarily show fader value on respective scribble when moving faders
/$ctl/layer	N			Layer node
/$ctl/layer/L	N			Left layer node

/$ctl/layer/L/sel	I	1..22	1..9 settable[29]
			10..22 fixed/pre-assigned
				Left layer select[30]
				`1: Ch 1..Ch 12`
				`2: Ch 13..Ch 24`
				`3: Ch 25..Ch 36`

[29] Full size console has 7 layers whereas Compact console has 9

[30] Showing full-size console layers names here

				4: Ch 36..Ch 40 / Aux 1..Aux 8 5: Bus 1..Bus 12 6: User 1 7: User 2 8: No-op 9: No-op 10: Ch 1..Ch 8 11: Ch 9..Ch 16 12: Ch 17..Ch 24 13: Ch 25..Ch 32 14: Ch 33..Ch 40 15: Aux 1..Aux 8 16: Bus 1..Bus 8 17: Bus 9..Bus 16 18: Main 1..Main 4 19: Matrix 1..Matrix 8 20: DCA 1..DCA 8 21: DCA 9..DCA 16 22: spilled layer
/$ctl/layer/L/$spidx	I	0..78		Internal parameter for spilled group 0: OFF 1..16: DCA 1..16 17-32: FX 1..16 … 61..76: BUS 1..16 77: AUTOX 78: AUTOY
/$ctl/layer/L/1	N	1..9		Left layer 1 node (see above)
/$ctl/layer/L/1/ofs	I	0..12		Left layer 1 offset (from <4 or4> keys for ex.)
/$ctl/layer/L/1/name	S		10 chars max CH1-12, CH13-24, CH25-36, CH37-AUX, BUSES, USER1, USER2	Left layer 1 name
/$ctl/layer/L/1/1	N	1..24		Left layer 1, node 1
/$ctl/layer/L/1/1/type	S		OFF, CH, BUS, DCA, MIDI, SEND, FX	Left layer 1, node 1 type (OSC patterns in italic below correspond to MIDI type)
/$ctl/layer/L/1/1/i	I	1..127		Left layer 1, node 1 index
/$ctl/layer/L/1/1/dst	I	1..16		Left layer 1, node 1 destination index (used for type SEND)
/$ctl/layer/L/1/1/val	*I*	*0..127*		*Left layer 1, node 1 value (when type MIDI)*
/$ctl/layer/C	N			Center layer node
/$ctl/layer/C/sel	I	1..22	1..6 settable 7..9 No-op 10..22 fixed/pre-assigned	Center layer select[31]: 1: DCA 2: Main/Matrix 3: Aux/FX 4: Bus/Master 5: User 1 6: User 2 7: No-op 8: No-op 9: No-op 10: Ch 1..Ch 8 11: Ch 9..Ch 16

[31] Showing full-size console layers names here

				12: Ch 17..Ch 24 13: Ch 25..Ch 32 14: Ch 33..Ch 40 15: Aux 1..Aux 8 16: Bus 1..Bus 8 17: Bus 9..Bus 16 18: Main 1..Main 4 19: Matrix 1..Matrix 8 20: DCA 1..DCA 8 21: DCA 9..DCA 16 22: spilled layer
/$ctl/layer/C/$spidx	I	0..76		Internal parameter for spilled group 0: OFF 1..16: DCA 1..16 17-32: FX 1..16 … 61..76: BUS 1..16 77: AUTOX 78: AUTOY
/$ctl/layer/C/1	N	1..9		Center layer 1 node (see above)
/$ctl/layer/C/1/ofs	I	0..8		Center layer 1 offset
/$ctl/layer/C/1/name	S		10 chars max DCA, MAIN, AUX, BUSES, USER1, USER2	Center layer 1 name
/$ctl/layer/C/1/1	N	1..16		Center layer 1, node 1
/$ctl/layer/C/1/1/type	S		OFF, CH, BUS, DCA, MIDI, SEND, FX	Center layer 1, node 1 type (OSC patterns in italic below correspond to MIDI type)
/$ctl/layer/C/1/1/i	I	1..127		Center layer 1, node 1 index
/$ctl/layer/C/1/1/dst	I	1..16		Left layer 1, node 1 destination index (used for type SEND)
/$ctl/layer/C/1/1/val	*I*	*0..127*		*Center layer 1, node 1 value (when type MIDI)*
/$ctl/layer/R	N			Right layer node
/$ctl/layer/R/sel	I	1..22	1..7 settable 8, 9 No-op 10..22 fixed/pre-assigned	Right layer select[32]: 1: Main 2: DCA 3: Channels 4: Aux/FX 5: Bus/Master 6: User 1 7: User 2 8: No-op 9: No-op 10: Ch 1..Ch 4 11: Ch 9..Ch 12 12: Ch 17..Ch 20 13: Ch 25..Ch 28 14: Ch 33..Ch 36 15: Aux 1..Aux 4 16: Bus 1..Bus 4 17: Bus 9..Bus 12 18: Main 1..Main 4 19: Matrix 1..Matrix 4 20: DCA 1..DCA 4 21: DCA 9..DCA 12

[32] Showing full-size console layers names here

				22: spilled layer
/$ctl/layer/R/$spidx	I	0..76		Internal parameter for spilled group 0: OFF 1..16: DCA 1..16 17-32: FX 1..16 … 61..76: BUS 1..16 77: AUTOX 78: AUTOY
/$ctl/layer/R/1	N	1..9		Right layer 1 node (see above)
/$ctl/layer/R/1/ofs	I	0..15		Right layer 1 offset
/$ctl/layer/R/1/name	S		MAIN, DCA, CH1-40, AUX, BUSES, USER1, USER2	Right layer 1 name
/$ctl/layer/R/1/1	N	1..16 (40 for…/R/3…)		Right layer 1, node 1. 16 nodes except for type CH1-40: 40 nodes
/$ctl/layer/R/1/1/type	S		OFF, CH, BUS, DCA, MIDI, SEND, FX	Right layer 1, node 1 type (OSC patterns in italic below correspond to MIDI type)
/$ctl/layer/R/1/1/i	I	0..127		Right layer 1, node 1 index
/$ctl/layer/R/1/1/dst	I	1..16		Right layer 1, node 1 destination index (used for type SEND)
/$ctl/layer/R/1/1/val	*I*	*0..127*		*Right layer 1, node 1 value (when type MIDI)*
/$ctl/user	N			User node
/$ctl/user/sel	I	1..16		User select[33]
/$ctl/user/mode	S		USER, 2TRK, WLIVE, MGRP, SHOW	User button mode (5 buttons above the wheel)
/$ctl/user/cmode	S		HA, GATE, COMP, FLT, U1, U2, U3, PAN	User channel mode (8 buttons top right corner of the console)
/$ctl/user/gpio	N			User GPIO node
/$ctl/user/gpio/1	N	1..4		User GPIO 1 node
/$ctl/user/gpio/1/bu	N			User GPIO 1 up node
/$ctl/user/gpio/1/bu/mode	S		OFF, MUTE, INS1, INS2, MGRP, DCAMUTE, SOF, SPILL, FXPAR, DAWBTN, DAWENC, CHPAGE, PAGE, FDRPAGE, VIEWPAGE, OTHER, GPIO, FSTART, SHOWCTL, SCENES, MIDICCT, MIDICCP, MIDINT, MIDINP, MIDIPGM, USBPR, SDRECA, SESSIONA, MARKERA, SDRECB, SESSIONB, MARKERB	User GPIO 1 up function (see appendix on buttons)
/$ctl/user/gpio/1/bu/name	S		16 chars max	User GPIO 1 up name (use a leading '|' to invert characters)
/$ctl/user/gpio/1/bu/$fname	S		16 chars max	User GPIO 1 up $fname [RO]
/$ctl/user/user	N			User Layer node (bottom with Link enabled)

[33] Setting values using the range 1..16, reported values are in the range 0..15

/$ctl/user/user/1	N	1..4		User layer button 1 node
/$ctl/user/user/1/bu	N			User layer button 1 upper row node
/$ctl/user/user/1/bu/mode	S		OFF, MUTE, INS1, INS2, MGRP, DCAMUTE, SOF, SPILL, FXPAR, DAWBTN, DAWENC, CHPAGE, PAGE, FDRPAGE, VIEWPAGE, OTHER, GPIO, FSTART, SHOWCTL, SCENES, MIDICCT, MIDICCP, MIDINT, MIDINP, MIDIPGM, USBPR, SDRECA, SESSIONA, MARKERA, SDRECB, SESSIONB, MARKERB	User layer button1 upper row function (see appendix on buttons)
/$ctl/user/user/1/bu/name	S		16 chars max	User layer button 1 upper row name (use a leading '|' to invert characters)
/$ctl/user/user/1/bu/$fname	S		16 chars max	User layer button 1 upper row function name [RO]
/$ctl/user/user/1/bd	N			User layer button 1 lower row node
/$ctl/user/user/1/bd/mode	S		OFF, MUTE, INS1, INS2, MGRP, DCAMUTE, SOF, SPILL, FXPAR, DAWBTN, DAWENC, CHPAGE, PAGE, FDRPAGE, VIEWPAGE, OTHER, GPIO, FSTART, SHOWCTL, SCENES, MIDICCT, MIDICCP, MIDINT, MIDINP, MIDIPGM, USBPR, SDRECA, SESSIONA, MARKERA, SDRECB, SESSIONB, MARKERB	User layer button 1 lower row function (see appendix on buttons)
/$ctl/user/user/1/bd/name	S		16 chars max	User layer button 1 lower row name (use a leading '|' to invert characters)
/$ctl/user/user/1/bd/$fname	S		16 chars max	User layer button 1 lower row function name [RO]
/$ctl/user/daw1	N			User DAW1 node
/$ctl/user/daw1/1	N	1..4		User DAW1 button 1 node
/$ctl/user/daw1/1/bu	N			User DAW1 button 1 upper row node
/$ctl/user/daw1/1/bu/mode	S		OFF, MUTE, INS1, INS2, MGRP, DCAMUTE, SOF, SPILL, FXPAR, DAWBTN, DAWENC, CHPAGE, PAGE, FDRPAGE, VIEWPAGE, OTHER, GPIO, FSTART, SHOWCTL, SCENES, MIDICCT, MIDICCP, MIDINT, MIDINP, MIDIPGM, USBPR, SDRECA, SESSIONA, MARKERA, SDRECB, SESSIONB, MARKERB	User DAW1 button 1 upper row function (see appendix on buttons)
/$ctl/user/daw1/1/bu/name	S		16 chars max	User DAW1 button 1 upper row name (use a leading '|' to invert characters)
/$ctl/user/daw1/1/bu/$fname	S		16 chars max	User DAW1 button 1 upper row function name [RO]

/$ctl/user/daw1/1/bd	N			User DAW1 button 1 lower row node
/$ctl/user/daw1/1/bd/mode	S		OFF, MUTE, INS1, INS2, MGRP, DCAMUTE, SOF, SPILL, FXPAR, DAWBTN, DAWENC, CHPAGE, PAGE, FDRPAGE, VIEWPAGE, OTHER, GPIO, FSTART, SHOWCTL, SCENES, MIDICCT, MIDICCP, MIDINT, MIDINP, MIDIPGM, USBPR, SDRECA, SESSIONA, MARKERA, SDRECB, SESSIONB, MARKERB	User DAW1 button 1 lower row function (see appendix on buttons)
/$ctl/user/daw1/1/bd/name	S		16 chars max	User DAW1 button 1 lower row name (use a leading '|' to invert characters)
/$ctl/user/daw1/1/bd/$fname	S		16 chars max	User DAW1 button 1 lower row function name [RO]
/$ctl/user/daw2	N			User DAW2 node
/$ctl/user/daw2/1	N	1..4		User DAW2 button 1 node
/$ctl/user/daw2/1/bu	N			User DAW2 button 1 upper row node
/$ctl/user/daw2/1/bu/mode	S		OFF, MUTE, INS1, INS2, MGRP, DCAMUTE, SOF, SPILL, FXPAR, DAWBTN, DAWENC, CHPAGE, PAGE, FDRPAGE, VIEWPAGE, OTHER, GPIO, FSTART, SHOWCTL, SCENES, MIDICCT, MIDICCP, MIDINT, MIDINP, MIDIPGM, USBPR, SDRECA, SESSIONA, MARKERA, SDRECB, SESSIONB, MARKERB	User DAW2 button 1 upper row function (see appendix on buttons)
/$ctl/user/daw2/1/bu/name	S		16 chars max	User DAW2 button 1 upper row name (use a leading '|' to invert characters)
/$ctl/user/daw2/1/bu/$fname	S		16 chars max	User DAW2 button 1 upper row function name [RO]
/$ctl/user/daw2/1/bd	N			User DAW2 button 1 lower row node
/$ctl/user/daw2/1/bd/mode	S		OFF, MUTE, INS1, INS2, MGRP, DCAMUTE, SOF, SPILL, FXPAR, DAWBTN, DAWENC, CHPAGE, PAGE, FDRPAGE, VIEWPAGE, OTHER, GPIO, FSTART, SHOWCTL, SCENES, MIDICCT, MIDICCP, MIDINT, MIDINP, MIDIPGM, USBPR, SDRECA, SESSIONA, MARKERA, SDRECB, SESSIONB, MARKERB	User DAW2 button 1 lower row function (see appendix on buttons)
/$ctl/user/daw2/1/bd/name	S		16 chars max	User DAW2 button 1 lower row name (use a leading '|' to invert characters)
/$ctl/user/daw2/1/bd/$fname	S		16 chars max	User DAW2 button 1 lower row function name [RO]

/$ctl/user/daw3	N			User DAW3 node
/$ctl/user/daw3/1	N	1..4		User DAW3 button 1 node
/$ctl/user/daw3/1/bu	N			User DAW3 button 1 upper row node
/$ctl/user/daw3/1/bu/mode	S		OFF, MUTE, INS1, INS2, MGRP, DCAMUTE, SOF, SPILL, FXPAR, DAWBTN, DAWENC, CHPAGE, PAGE, FDRPAGE, VIEWPAGE, OTHER, GPIO, FSTART, SHOWCTL, SCENES, MIDICCT, MIDICCP, MIDINT, MIDINP, MIDIPGM, USBPR, SDRECA, SESSIONA, MARKERA, SDRECB, SESSIONB, MARKERB	User DAW3 button 1 upper row function (see appendix on buttons)
/$ctl/user/daw3/1/bu/name	S		16 chars max	User DAW3 button 1 upper row name (use a leading '|' to invert characters)
/$ctl/user/daw3/1/bu/$fname	S		16 chars max	User DAW3 button 1 upper row function name [RO]
/$ctl/user/daw3/1/bd	N			User DAW3 button 1 lower row node
/$ctl/user/daw3/1/bd/mode	S		OFF, MUTE, INS1, INS2, MGRP, DCAMUTE, SOF, SPILL, FXPAR, DAWBTN, DAWENC, CHPAGE, PAGE, FDRPAGE, VIEWPAGE, OTHER, GPIO, FSTART, SHOWCTL, SCENES, MIDICCT, MIDICCP, MIDINT, MIDINP, MIDIPGM, USBPR, SDRECA, SESSIONA, MARKERA, SDRECB, SESSIONB, MARKERB	User DAW3 button 1 lower row function (see appendix on buttons)
/$ctl/user/daw3/1/bd/name	S		16 chars max	User DAW3 button 1 lower row name (use a leading '|' to invert characters)
/$ctl/user/daw3/1/bd/$fname	S		16 chars max	User DAW3 button 1 lower row function name [RO]
/$ctl/user/daw4	N			User DAW4 node
/$ctl/user/daw4/1	N			User DAW4 button 1 node
/$ctl/user/daw4/1/bu	N	1..4		User DAW4 button 1 upper row node
/$ctl/user/daw4/1/bu/mode	S		OFF, MUTE, INS1, INS2, MGRP, DCAMUTE, SOF, SPILL, FXPAR, DAWBTN, DAWENC, CHPAGE, PAGE, FDRPAGE, VIEWPAGE, OTHER, GPIO, FSTART, SHOWCTL, SCENES, MIDICCT, MIDICCP, MIDINT, MIDINP, MIDIPGM, USBPR, SDRECA, SESSIONA, MARKERA, SDRECB, SESSIONB, MARKERB	User DAW4 button 1 upper row function (see appendix on buttons)
/$ctl/user/daw4/1/bu/name	S		16 chars max	User DAW4 button 1 upper row name (use a leading '|' to invert characters)

/$ctl/user/daw4/1/bu/$fname	S		16 chars max	User DAW4 button 1 upper row function name [RO]
/$ctl/user/daw4/1/bd	N			User DAW4 button 1 lower row node
/$ctl/user/daw4/1/bd/mode	S		OFF, MUTE, INS1, INS2, MGRP, DCAMUTE, SOF, SPILL, FXPAR, DAWBTN, DAWENC, CHPAGE, PAGE, FDRPAGE, VIEWPAGE, OTHER, GPIO, FSTART, SHOWCTL, SCENES, MIDICCT, MIDICCP, MIDINT, MIDINP, MIDIPGM, USBPR, SDRECA, SESSIONA, MARKERA, SDRECB, SESSIONB, MARKERB	User DAW4 button 1 lower row function (see appendix on buttons)
/$ctl/user/daw4/1/bd/name	S		16 chars max	User DAW4 button 1 lower row name (use a leading '|' to invert characters)
/$ctl/user/daw4/1/bd/$fname	S		16 chars max	User DAW4 button 1 lower row function name [RO]
/$ctl/user/1	N	1..16		User 1 node
/$ctl/user/1/1	N	1..4		User 1 button/encoder 1 node
/$ctl/user/1/1/led	I	0..1		User 1 LED 1 off/on switch
/$ctl/user/1/1/col	I	1..12		User 1 LED 1 color
/$ctl/user/1/1/enc	N			User 1 encoder 1 node
/$ctl/user/1/1/enc/mode	S		OFF, FDR, PAN, DCA, SSND, FSND, FX, DAWMCU, MIDICC, SD A, SD B	User 1 encoder 1 function (see appendix on buttons)
/$ctl/user/1/1/enc/name	S		16 chars max	User 1 encoder 1 name (use a leading '|' to invert characters)
/$ctl/user/1/1/enc/$fname	S		16 chars max	User 1 encoder 1 function name [RO]
/$ctl/user/1/1/bu	N			User 1 button 1 upper row node
/$ctl/user/1/1/bu/mode	S		OFF, MUTE, INS1, INS2, MGRP, DCAMUTE, SOF, SPILL, FXPAR, DAWBTN, DAWENC, CHPAGE, PAGE, FDRPAGE, VIEWPAGE, OTHER, GPIO, FSTART, SHOWCTL, SCENES, MIDICCT, MIDICCP, MIDINT, MIDINP, MIDIPGM, USBPR, SDRECA, SESSIONA, MARKERA, SDRECB, SESSIONB, MARKERB	User 1 button 1 upper row function (see appendix on buttons)
/$ctl/user/1/1/bu/name	S		16 chars max	User 1 button 1 upper row name (use a leading bu/mode'|' to invert characters)
/$ctl/user/1/1/bu/$fname	S		16 chars max	User 1 button 1 upper row function name [RO]
/$ctl/user/1/1/bd	N			User 1 button 1 lower row node
/$ctl/user/1/1/bd/mode	S		OFF, MUTE, INS1, INS2, MGRP, DCAMUTE, SOF, SPILL, FXPAR, DAWBTN,	User 1 button 1 lower row function (see appendix on buttons)

			DAWENC, CHPAGE, PAGE, FDRPAGE, VIEWPAGE, OTHER, GPIO, FSTART, SHOWCTL, SCENES, MIDICCT, MIDICCP, MIDINT, MIDINP, MIDIPGM, USBPR, SDRECA, SESSIONA, MARKERA, SDRECB, SESSIONB, MARKERB	
/$ctl/user/1/1/bd/name	S		16 chars max	User 1 button 1 lower row name (use a leading '|' to invert characters)
/$ctl/user/1/1/bd/$fname	S		16 chars max	User 1 button 1 lower row function name [RO]
/$ctl/user/cuser	N	1..3		Cuser node
/$ctl/user/cuser/1	S		1, 2, 3, … , 15, 16	Cuser 1 rotary knob position; Keeps the bus send value assigned to respective channel for the F1..F3 section. Can be set/changed by holding the F1..F3 key and turning the knob.
/$ctl/gpio	N			GPIO node
/$ctl/gpio/1	N	1..4		GPIO 1 node
/$ctl/gpio/1/mode	S		TGLNO, TGLNC, INNO, INNC, OUTNO, OUTNC	GPIO 1 mode (TGL: toggle; NO: normally open; NC: normally closed)
/$ctl/gpio/1/$state	I	0..1		GPIO 1 state [RO]
/$ctl/gpio/1/gpstate	I	0..1		GPIO 1 gpio state
/$ctl/safes	N			Global Safes node
/$ctl/safes/ch	S		40 chars max	Ch safes switches (+ or space)
/$ctl/safes/aux	S		8 chars max	Aux safes switches (+ or space)
/$ctl/safes/bus	S		16 chars max	Bus safes switches (+ or space)
/$ctl/safes/main	S		4 chars max	Main safes switches (+ or space)
/$ctl/safes/mtx	S		8 chars max	Matrix safes switches (+ or space)
/$ctl/safes/dca	S		16 chars max	DCA safes switches (+ or space)
/$ctl/safes/mute	S		8 chars max	Mute safes switches (+ or space)
/$ctl/safes/fx	S		16 chars max	FX safes switches (+ or space)
/$ctl/safes/source	N			Source Safes node
/$ctl/safes/source/LCL	S		24 chars max	LCL source safes switches (+ or space)
/$ctl/safes/source/AUX	S		8 chars max	AUX source safes switches (+ or space)
/$ctl/safes/source/A	S		48 chars max	A source safes switches (+ or space)
/$ctl/safes/source/B	S		48 chars max	B source safes switches (+ or space)
/$ctl/safes/source/C	S		48 chars max	C source safes switches (+ or space)
/$ctl/safes/source/SC	S		32 chars max	SC source safes switches (+ or space)
/$ctl/safes/source/USB	S		48 chars max	USB source safes switches (+ or space)
/$ctl/safes/source/CRD	S		64 chars max	CRD source safes switches (+ or space)
/$ctl/safes/source/MOD	S		64 chars max	MOD source safes switches (+ or space)
/$ctl/safes/source/PLAY	S		4 chars max	REC source safes switches (+ or space)
/$ctl/safes/source/AES	S		2 chars max	AES source safes switches (+ or space)
/$ctl/safes/source/USR	S		48 chars max	USR source safes switches (+ or space)
/$ctl/safes/source/OSC	S		2 chars max	Osc source safes switches (+ or space)
/$ctl/safes/output	N			Output Safes node
/$ctl/safes/output/LCL	S		8 chars max	LCL out safes switches (+ or space)

/$ctl/safes/output/AUX	S		8 chars max	AUX out safes switches (+ or space)
/$ctl/safes/outpu/A	S		48 chars max	A out safes switches (+ or space)
/$ctl/safes/output/B	S		48 chars max	B out safes switches (+ or space)
/$ctl/safes/output/C	S		48 chars max	C out safes switches (+ or space)
/$ctl/safes/output/SC	S		32 chars max	SC out safes switches (+ or space)
/$ctl/safes/output/USB	S		48 chars max	USB out safes switches (+ or space)
/$ctl/safes/output/CRD	S		64 chars max	CRD out safes switches (+ or space)
/$ctl/safes/output/MOD	S		64 chars max	MOD out safes switches (+ or space)
/$ctl/safes/output/REC	S		4 chars max	REC out safes switches (+ or space)
/$ctl/safes/output/AES	S		2 chars max	AES out safes switches (+ or space)
/$ctl/safes/area	N			Area safes node
/$ctl/safes/area/LEFT	S		9 chars max	Left area safes switches (+ or space)
/$ctl/safes/area/CENTER	S		9 chars max	Center area safes switches (+ or space)
/$ctl/safes /area/RIGHT	S		9 chars max	Right area safes switches (+ or space)
/$ctl/safes/custom	S		22 chars max	Custom area safes switches (+ or space)
/$ctl/safes/setup	S		3 chars max	Setup area safes switches (+ or space)
/$ctl/daw	N			DAW node
/$ctl/daw/on	I	0..1		DAW enable
/$ctl/daw/conn	S		DIN, USB	DAW connection
/$ctl/daw/emul	S		MCU, HUI	DAW emulation
/$ctl/daw/config	S		CC, MSTR, MSTR1EXT, MSTR2EXT	DAW configuration
/$ctl/daw/ccup	I	0..1		DAW use upper cc
/$ctl/daw/disjog	I	0..1		DAW disable wheel during play
/$ctl/daw/preset	S		-, cubase, live, logicx, nuendo, protools, reaper, studioone	DAW last loaded preset
/$ctl/daw /$on	I	0..1		DAW enable switch
/$ctl/daw/$bpage	I	0..4		DAW on button page
/$ctl/daw/$btntouch	I	0..1		DAW on button sel fader touch
/$ctl/daw/$btnvpot	I	0..1		DAW on button sel vpot
/$ctl/daw/$btnrecrdy	I	0..1		DAW on button sel record ready
/$ctl/daw/$btnauto	I	0..1		DAW on button sel auto
/$ctl/daw/$btnvsel	I	0..1		DAW on button sel v-sel
/$ctl/daw/$btninsert	I	0..1		DAW on button sel insert
/$ctl/midi	N			Midi node
/$ctl/midi/enchctl	S		OFF, DIN, USB	Channel control (FDR, MUTE, PAN)
/$ctl/midi/enfxctl	S		OFF, DIN, USB	FX parameter control
/$ctl/midi/encustctl	S		OFF, DIN, USB	Custom control (RX only)
/$ctl/midi/ensysex	S		OFF, DIN, USB	SYSEX control
/$ctl/midi/enmidicc	S		OFF, DIN, USB	External MIDI control
/$ctl/midi/enscenes	S		OFF, DIN, USB	Scene change
/$ctl/midi/enshowctl	S		OFF, DIN, USB	Show control
/$ctl/midi/enscenetx	S		OFF, DIN, USB	Scene MIDI TX
/$ctl/OSC	N			OSC node
/$ctl/OSC/ronly	I	0..1		Console OSC read only switch
/$ctl/lib	N			Library node (Shows, Scenes, Snaps, …)

/$ctl/lib/$scenes	S		Ex: `scene_1, scene_2, scene_3`	List of scene[34] names [RO] [35] in the currently opened show
/$ctl/lib/$actidx	I	0..n		Scene number currently loaded/active [RO][36]
/$ctl/lib/$active	S		256 chars max	Name of the active scene [RO], ex: `I:SHOW2/scene_1.snap`
/$ctl/lib/$actshow	S		256 chars max	Name of the active show [RO], after having pressed on the "OPEN SHOW" icon, ex: `I:SHOW2`
/$ctl/lib/$action	S		IDLE, GOPREV, GONEXT, GO, PREV, NEXT, GOTAG	Show control actions, after having pressed on the "OPEN SHOW" icon
/$ctl/lib/$actionidx	I	0..16384		Scene index user selection, in the list of Scenes. A show must be opened. Use `/$ctl/lib/$action ,s GO` to load the Scene entity `$actionidx` points to
/$ctl/$globals	N			CTL Global Settings node
/$ctl/$globals/fdrsel	I	0..1		Screen Touch Fader Select
/$ctl/$globals/fdrres	S		NORM, FINE, AUTO	Fader resolution
/$ctl/$globals/fdrspd	S		SLOW, MED, FAST	Fader speed
/$ctl/$globals/mousetchdis	I	0..1		Mouse disable touch
/$ctl/$globals/mousespd	F	0.1..2.0	191 steps	Mouse speed
/$ctl/$globals/tapflash	S		OFF, 8X, ON	Tap Tempo Flash
/$ctl/$globals/srcdisp	I	0..1		Show source on scribble
/$ctl/$globals/lockmtr	I	0..1		Show meter page when locked
/$ctl/$globals/cf_load	I	0..1		Confirm Snapshot Load
/$ctl/$globals/cf_upd	I	0..1		Confirm Snapshot Update
/$ctl/$globals/usewheel	I	0..1		Use wheel to navigate in lists of items (snaps, files,…)
/$ctl/$globals/$filesort	S		A→Z, Z→A, 0→9, 9→0	File sort order
/$ctl/$globals/$noautosave	I	0..1		Auto save switch (0=autosave)
/$ctl/$globals/$savenow	I	0..1		Save console data now[37]

[34] A Scene is a Snap, a Snippet, a Preset, an Audio clip, or a combination thereof, referenced in a Show file

[35] Only the first name of the list is returned by std OSC command. You must use the node definition command (OSC or native interface) to get the full contents, for ex: `/$ctl/lib~~~,s~~?~~~.` A show must be opened for the command to be active

[36] A show must be opened for the command to be active. 0 means "no show Scene loaded"

[37] This command should not be used as part of a program loop as it will wvwntually wear the flash memory where data is saved. This command may change in the future.

WING native / binary data interface

WING native / binary data interface

WING exposes a binary structure that contains all data presented above in the WING snapshot JSON structure chapter, and more objects. Some objects are `READ ONLY` while others can bet accessible for `get()` and `set()` functions. One can access the object data, effectively getting access to the value assigned to the object, or to the object description, a special class that provides the object name, type, min/max values, or enumerated values.
As mentioned before, communication takes place using `TCP`. We give all the basic/necessary details for communicating with WING below; this data is coming from Behringer.
Applications can communicate with the console using `TCP` port `2222`. The console will reject further connection requests, if the maximum number of simultaneous connections (currently `16`) is reached. Open connections will time out after `10 seconds` of inactivity (on the receiving side).

Communication Channels

Communication uses 14 distinct *channels* [1..14] corresponding to the Channel IDs presented below

Table 1. Channel Usage

channel	ChID	Usage
1	0	n/u
2	1	Audio Engine & Control requests
3	2	n/u
4	3	Meter Data Requests
5	4	n/u
6	5	n/u
7	6	n/u
8	7	n/u
9	8	n/u
10	9	n/u
11	A	n/u
12	B	n/u
13	C	n/u
14	D	n/u

To select/change the active channel, use the following sequence
`0xdf, 0xd<ChID>`

When communicating with WING, the escape byte `0xdf` should be handled carefully, as shown in the two routines shown below for sending and receiving data.

Sample receive routine

```
#define NRP_ESCAPE_CODE         0xdf
#define NRP_CHANNEL_ID_BASE     0xd0
#define NRP_NUM_CHANNELS        14

void Cnrpclientconnector::nrpc_data_rx(byte db)
{
  if (db == NRP_ESCAPE_CODE && !escf) escf = true;
  else {
    if (escf) {
      if (db != NRP_ESCAPE_CODE) {
        escf = false;
        if (db == NRP_ESCAPE_CODE - 1) db = NRP_ESCAPE_CODE;
        else if (db >= NRP_CHANNEL_ID_BASE &&
                       db < NRP_CHANNEL_ID_BASE + NRP_NUM_CHANNELS) {
          if (ch_id_rx != db - NRP_CHANNEL_ID_BASE) {
            ch_id_rx = db - NRP_CHANNEL_ID_BASE;
          }
          return;
        } else if (ch_id_rx >= 0) data_rx(ch_id_rx, NRP_ESCAPE_CODE);
      }
    }
    if (ch_id_rx >= 0) data_rx(ch_id_rx, db);
  }
}
```

Example: The sequence `D702DFDEAF0E02` will in fact represent `D702DFAF0E02`

Sample transmit routine

```
void Cnrpclientconnector::data_tx(int ch_id, const void* data, int len)
{
  assert(ch_id >= 1 && ch_id <= NRP_NUM_CHANNELS);

  if (ch_id_tx != ch_id) {
    ch_id_tx = ch_id;
    nrpc_data_tx_flush();
    nrpc_data_tx(NRP_ESCAPE_CODE);
    nrpc_data_tx(ch_id + NRP_CHANNEL_ID_BASE);
  }

  bool esc = false;
  byte* dpp = (byte*)data;
  while (len-- > 0) {
    byte db = *dpp++;
    if (db == NRP_ESCAPE_CODE) esc = true;
    else {
      if (esc && db >= NRP_CHANNEL_ID_BASE &&
                  db <= NRP_CHANNEL_ID_BASE + NRP_NUM_CHANNELS) {
        db = NRP_ESCAPE_CODE - 1;
        dpp--;
        len++;
      }
      esc = false;
    }
    nrpc_data_tx(db);
  }
  if (esc) nrpc_data_tx(NRP_ESCAPE_CODE - 1);
}
```

Examples: With current tx channel being 3, sending `d702dfaf0e02` to channel 1 will transfer `dfd0d702dfaf0e02`, sending `d702dfdf0e02` to channel 2 will transfer `dfd1d702dfdf0e02`, and sending `d702dfd10e02` to channel 1 again will transfer `dfd0d702dfded10e02`

Channel 2: Audio Engine

Communication with the audio engine uses a token/data based binary stream protocol. Words are 2 bytes (big endian), longs are 4 bytes (big endian).

Binary Stream Format

Table 2. Binary Stream Protocol Tokens

Token	Data	Function
0x00		false; off; 0
0x01		true; on; 1
0x02…0x3f		int 2…63
0x40…0x7f		node index 1…64
0x80…0xbf		string[1…64]
0xc0…0xcf		node name[1…16]
0xd0		empty string
0xd1	byte	string[1…256]
0xd2	word	node index 1…65536
0xd3	word	int16
0xd4	long	int32
0xd5	long	float32
0xd6	long	raw float32 (*0.0*…*1.0*)
0xd7	long	node hash
0xd8		click (toggle)
0xd9	byte	step (inc/dec)
0xda		node tree: goto root node
0xdb		node tree: 1 level up
0xdc		data request
0xdd		request node definition (current node)
0xde		end of data/def request
0xdf	word	node definition response (word: data length in bytes)
0xe0…0xff		not used

Navigation within the nodes tree can be done by using `root, 1 level up, node index, _node name,` or `node hash` tokens.
Definitions of all `subnodes` within the current `node` can be requested with the `request node definition` token.

Node Definition Response

```
0xdf, len.w, [len.l], parent.l, hash.l, index.w, namelen.b, name, longnamelen.b, longname, flags.w,
        (If len.w==0 then len.l is used)
        node:   -
        linf:   min.f, max.f, steps.l
        logf:   min.f, max.f, steps.l
        fdr:    -
        int:    min.l, max.l
        enum:   count.w, [itemlen.b, item, longitemlen.b, longitem] * count
        fenum:  count.w, [itemval.f, longitemlen.b, longitem] * count
        string: maxlen.w
```

Where the `node type` can be derived from the `flags` as follows:

Table 3. Node Flags

Flags	Function	Details
b0…b3	unit	0: none 1: dB 2: % 3: ms 4: Hz 5: mtrs 6: seconds 7: octaves
b4…b7	type	0: node 1: lin float 2: log float

		3: fader level 4: integer 5: string enum 6: float enum 7: string
b9	r/o	read-only flag

Channel 3: Metering

Use this channel to request metering data from the console. The data is sent back to the client `IP` to the specified `port`. Meter data times out after `5` seconds.
Each metering data block is prefixed by a client-specified 4byte `report id`.
In order to avoid confusion on the receiver side, the client should use different report ids for different meter collections. Meter collections are specified by using multiple type/index elements.
Specified meters are sent back in the sequence corresponding to the request message.

Details:
Meter values are coded on `2 bytes` (signed, big endian). Level values are in `1/256` dB.

Most data are returned in `1/256` steps. This is typically the case for `Gate gain` and `Dyn gain` values. The returned value is multiplied/adjusted with a fixed value to cover for the plugin model data range in use. For most of them `1.0 (256)` maps to `20` dB gain reduction. Standard Wing gate is `60` dB.

`fx` meter subscriptions return 4 levels and 6 state meters (see table below: Meter Data). Band gain reduction is in the first 5 of these. `Value` in dB = return value `* 6.0 / 2048`.

`Gate LED` and `Dyn State` in * V2 subscriptions return a value of 0 or 1, depending on the state of the respective channel's Gate and Dyn LEDs, respectively.

Meter Request Tokens

Table 4. Meter Request Tokens

Token	Data	Function
0xd3	word	client *UDP* port (2 bytes big endian; set at least once)
0xd4	long	report id (repeat this token with current id to reset transmission timeout)
0xdc		start of meter collection definition
	0xa0	channel (1…40)
	0xa1	aux (1…8)
	0xa2	bus (1…16)
	0xa3	main (1…4)
	0xa4	matrix (1…8)
	0xa5	dca (1…8)
	0xa6	fx processor (1…16)
	0xa7	source (input) device (1…16)
	0xa8	output device (1…11)
	0xa9	monitor (no index)
	0xaa	rta (no index)
	0xab	channel V2 (1…40)
	0xac	aux V2 (1…8)
	0xad	bus V2 (1…16)
	0xae	main V2 (1…4)
	0xaf	matrix V2 (1…8)
	0x00…0x7f	index 1…128 (can be repeated multiple times)
0xde		end of meter collection definition

Example specification

```
0xdc 0xa0 0x00 0x01 0x08 0xa6 0x04 0xde
```

Requests meter data for channels strips 1,2,9, and fx 5.

Meter Data Packet Structure := <report id (4 bytes)><Meter Data (n words)>

Meter Data

Table 5. Meter Data	
Section	Contents
channel aux bus main matrix	input left input right output left output right gate key gate gain dyn key dyn gain
dca	pre fader left pre fader right post fader left post fader right
fx	input left input right output left output right state meters (1…6)
source	source group levels (i.e. local ins: 8 meters)
output	output group levels (i.e. local outs: 8 meters)
monitor	solo bus left solo bus right mon 1 left mon 1 right mon 2 left mon 2 right
rta	rta slot meters (120)
Channel V2 aux V2 bus V2 main V2 matrix V2	input left input right output left output right gate key gate gain gate led dyn key dyn gain dyn state automix gain

We show below a typical binary communication sequence for requesting metering data:

```
→W  7 B: dfd3d33737dfd1                    // Declaring port to use is 0x3737 = 14135
→W 13 B: dfd3d400000002dca001dedfd1        // Meters request id = 2, channel 02
→W  9 B: dfd3d400000002dfd1                // Renew request id = 2 meter data for 5s
```

Introducing wapi (wapi)

The previous chapters on `JSON` structures and binary, token-based communication may not be very accessible to many programmers. For that reason, a more accessible API (Application Programming Interface) available as a set of include files and libraries is proposed here. It is written in C, which ensure it can easily be used in many applications, providing good performance.

There are two include files that should always be part of your programs as they contain information about the `JSON` structure and the objects respective binary pointers in WING. The API provides an abstraction layer to the binary interface and procedure calls for standard functions to get, set, manipulate WING data.

At the API level, WING data can be 32bit int, 32bit float or string data. All API data in little-endian, enabling easy use in standard programming languages.

Besides this document, the wapi API consists of two include files and a library:

`wapi.h` is the main include file containing enumerated types for errors, token types, and wapi abstraction enumerated tokens.

`wext.h` is a file containing the definitions of external library calls to wapi.lib the actual library of API functions.

`wapi.lib` is a static-link library for linking with your application. The library contains all wapi functions that can be used and are described later in this document.

A typical program accessing WING starts with an '*open*' function and ends with a '*close*' function. These two functions establish the communication path to WING on your local network and ensure data is properly cleaned when leaving the program.

Programs communicate with WING over network. The API call `wOpen()` is used to establish communication link between WING and the application.

WING supports multiple formats, including integers, floats, and strings types. The API will try to ensure conversions as best as possible in order to match the requested format either by WING or by the API command. For example, if you request float data from a WING token which is an integer, the API will convert the integer to float before returning the data. Similarly, if you set a WING token of type string by sending it a float value, the float data will be changed to string before being sent to WING.

wapi tokens

`wapi` makes use of `tokens` (an enumerated 32bits integer acting as a unique identifier) to identify the subtrees and leaves of the WING `JSON` hierarchy structure. WING tokens are easily identified by their name, based on their corresponding `JSON` structure name taken from the WING hierarchical data tree we already presented in this document.

For example the identifier for "channel 1 mute control", a.k.a "`ch.1.mute`" in the `JSON` tree is known as token `CH_1_MUTE`. The respective data in WING internal data structure is an integer that can be 0 or 1 and as written above, can be modified (or set) from integer, float or even string values, and can be returned to the application as an integer, a float of even a string. Following the naming convention above "channel 1 fader value" will be identified as "`CH_1_FDR`", "bus 14 panoramic value" will be identify as "`BUS_14_PAN`", and so on.

Some identifiers have names that are not as obvious, and this is due and to map to some of the dynamically assignable subtrees of the `JSON` tree. Typically, the filter, gate, compressor, and equalizer of WING channels can be assigned different plugin models, set for example using the `CH_1_EQ_MDL` (or `OSC ch/1/eq/mdl`) known as channel 1 EQ model in the case of channel 1 and its EQ setting. If you report to the different types (or models) of EQ plugins in the appendices of this document, you will see that the EQ model can be one of several choices, each model having different settings. In fact, every single setting maps to a given token, based on their respective listing number, e.i. `setting #7`, a.k.a. "`1q`" for EQ model "`STD`" will have the same token value as `setting #7` for EQ model `SOUL`, known as "`1mg`".

To enable `wapi` managing these different mapping, all `JSON` "dynamic" parameters are named after their listing number, rather than their name for a given effect or plugin model. As a result, and taking the case of EQ models "`STD`" and "`SOUL`" above, setting "`1q`" and "`1mg`" will have the share the same token ending with "`7`" (for listing #7).

The naming convention above applies to the following:

- Channel: filter, gate, compressor, equalizer ["`1`", "`2`", … "`32`"]
- Bus, Mains, Matrix: compressor, equalizer ["`1`", "`2`", … "`32`"]
- FX: all fx meter settings ["`1`", "`2`", … "`32`"]
- GPIOs ["`1`", "`2`"]
- User buttons ["`1`", "`2`", "`3`"]
- Layered user encoders and buttons ["`1`", "`2`", "`3`"]

Some tokens correspond to read-only data; trying to change their value will result in an error returned to the calling function.

WING tokens are listed in an include file: `wapi.h` that must be included in your program. The `wapi.h` include file also contains the status or error codes that can be returned by API function calls.

Compiling a program using wapi

All function calls are regrouped in a binary library: `wapi.lib`, that you must include at link time.
A typical compilation of a source file `wtest.c` in a Windows environment can be as follows:

```
gcc -O3 -Wall -c -fmessage-length=0 -o wtest.o "wtest.c"
gcc "-LC:<path to wapi.lib>" -o wtest.exe wtest.o -lwapi -lws2_32
```

Don't forget to set the correct path to the `wapi.lib` file in the above compilation/link lines!
Depending on your application, you may need to provide additional Windows dynamic libraries references (i.e. -lgdi32, -lcomdlg32, etc.)

WING parameters can be set (or modified) using the `wSetxxx` API family of calls; Similarly, the parameters can be retrieved from WING using the `wGetxxx` API family of calls. The following pages will present all API functions and will include examples of source code to help you in your first steps with wapi.
Additional calls will serve establishing the communication path with the console, and several services and utility functions needed to parse, or help with data.

wapi Reference Guide

wapi Reference Guide

Open and Close

Int wOpen(char wip)*

`wOpen()` initializes global variables for the application and opens the communication with a WING console responding at `IP` address wip.
`wip` is a string containing the console `IP` data in the form "`xxx.xxx.xxx.xxx`"; if the console `IP` address is unknown, `wip` should be an empty string and provide enough characters to store the IP address where WING will be found. The `wOpen()` function will attempt a network broadcast announce on the `/24`[38] of the local network to identify the first WING that will reply on the local network.
Upon successful completion the function will return `WSUCCESS` and if the `wip` parameter was an empty string when calling the function, it will contain the `IP` at which the console was found. Other values can be returned in case of issues or errors reported.
Once connection is established with the console, it will be kept active for about 10 seconds after which the console will close the link. The `wKeepAlive()` function can be called (before the desk closes communication) to extend the link active another 10 seconds.
It must be noted that if a connection is kept active, changes made directly at the console (by moving a fader, or pressing buttons for example) will generate data the application will continuously receive. This can represent a lot of data the application must be ready to accept and manage. It can also be the source of incorrect data returned to `Get` functions and specific care should be taken when developing live or event-driven applications.

void wClose()

`wClose()` ensures data is correctly disposed of when your program ends. It should be the last call before the `return` statement or `exit` call in your application.

int wVer()

`wVer()` returns the version of the wapi library file being used. The returned version is in the form 'major.minor.revision', and its value is provided as `0x0000MmmV`, with `M.mm` being the standard major.minor version number corresponding as close as possible to the Wing FW release wapi was based on, and `V` represents a software build revision number within M.mm.

[38] For example, 198.51.100.0/24 is the prefix of the Internet Protocol version 4 network starting at the given address, having 24 bits allocated for the network prefix, and the remaining 8 bits reserved for host addressing. Addresses in the range 198.51.100.0 to 198.51.100.255 belong to this network.

Setting Values

int wSetTokenFloat(wtoken token, float fval)

The `wSetTokenFloat()` sets WING token `token` to float value `fval`. If the token `token` is of a different type than float, `fval` will be adapted to the format expected by token `token`; if token `token` corresponds to a dynamic parameter named `1` to `32`, no format change will take place and the function will set token `token` using `fval` as float.
For example, sending value `444.0` to WING token `CH_1_PEQ_1F` will be sent as a 32bit float value. WING will nevertheless adjust it to the nearest valid value of `444.533997`. Sending that same value `444.0` to WING token `CH_1_PEQ_ON` will result in a setting to `1`; Finally, sending value `444.0` to WING token `CH_1_NAME` will change the channel name to `444.00`.
In order to change the value of `CH_1_EQ_1` to say 'ten' should be sent as float `10.0`, assuming the parameter's value expected type is float.
The function returns `WSUCCESS` if the requested operation was successful, other values can be returned, such as `WZERO` if no suitable format was found for adapting the value of `fval`, or `WSEND_TCP_ERROR` if an error took place while communicating with WING. Attempting to set a value on a token of type `NODE` will return `WNODE`.

int wSetTokenInt(wtoken token, int ival)

The `wSetTokenInt()` sets WING token `token` to int value `ival`. If the token `token` is of a different type than int, `ival` will be adapted to the format expected by token `token`; if token `token` corresponds to a dynamic parameter named `1` to `32`, no format change will take place and the function will set token `token` using `ival` as int.
For example, sending value `444` to WING token `CH_1_PEQ_ON` will result in a setting to `1`; Finally, sending integer value `444` to WING token `CH_1_NAME` will change the channel name to `444`.
In order to change the value of `CH_1_GATE_5` to say 'one' with `CH_1_GATE_MDL` set to "`9000G`" should be sent as int `1`, as the parameter's value expected type is int.
The function returns `WSUCCESS` if the requested operation was successful, other values can be returned, such as `WZERO` if no suitable format was found for adapting the value of `ival`, or `WSEND_TCP_ERROR` if an error took place while communicating with WING. Attempting to set a value on a token of type `NODE` will return `WNODE`.

int wSetTokenString(wtoken token, char str)*

The `wSetTokenString()` function takes as input a WING token `token` and a string `str`. It sends to WING the value of `str` after it has been adapted to the format expected by the WING token it is sent to.
For example, sending string "`444`" to WING token `CH_1_PEQ_1F` will be sent as a 32bit float value of `444.0`; WING will the adjust it to the nearest valid value of `444.533997`. Sending that same string **"`444`"** to WING token `CH_1_PEQ_ON` will result in a setting to `1`, Finally, sending string "`444`" to WING token `CH_1_NAME` will change the channel name to `444`.
In order to change the value of `CH_1_GATE_6` to say 'gate' with `CH_1_GATE_MDL` set to "`9000G`" should be sent as "`GATE`", as the parameter's value expected type is string.
The function returns `WSUCCESS` if the requested operation was successful, other values can be returned, such as `WZERO` if no suitable format was found for adapting the string `str`, or `WSEND_TCP_ERROR` if an error took place while communicating with WING. Attempting to set a value on a token of type `NODE` will return `WNODE`.

Getting Values

wapi offers several procedure calls to retrieve data from WING; specific datasets can be of use when getting data. These are defined in **wapi.h**:

wvalue is a union type definition to enable receiving several types of data in a single 32bits field.

```
typedef union {
    unsigned int       uval;   // unisgned integer type data
    int                ival;   // integert type data
    float              fval;   // float type data
    char*              sval;   // pointer to string
} wvalue;
```

wtype is an enumerated list of ints to provide the data type returned in **wvalue**.

```
typedef enum wtype {
    UNKN = -1,
    NODE,       // node type (unsigned int)
    I32,        // int type
    F32,        // float type
    S32,        // string (char*) type
    V32         // an unsigned int or void type to accept all the above
} wtype;
```

wTV is a C structure defined in the **wapi.h** file as follows:

```
typedef struct {
    wtoken    token;
    wtype     type;
    union {
        unsigned int udata;
        int          idata;
        float        fdata;
        char*        sdata;
    } d;
} wTV;
```

The filling of the structure is obvious for **int** and **float** data; **string** data sets are dynamically allocated and the pointer of the allocated string is saved in **sdata**; if the string returned from the console is an empty string, no memory allocation takes place and a **NULL** pointer is set in **sdata**.

wtype wGetType(wtoken token)

wGetType() returns the type associated to the token **token**; the returned value is one of the types listed in the **wtype** list described above.

char wGetName(wtoken token)*

wGetName() returns the string **JSON** descriptor corresponding to token **token**. The returned string is part of the constant definitions of **wapi** and cannot be altered by the calling application.
Note that the string "**$$unkown**" (not part of the **JSON** tree leaves) is returned for tokens that are not found.

whash wGetHash(wtoken token)

`wGetHash()` returns the binary descriptor (an unsigned int) corresponding to token `token`. The returned data can be used to identify specific entries in binary maps returned by `wapi` with the `wGetBinaryNode()` call (see later in this document). The value `0` is returned for tokens that are not found.

*int wGetToken(wtoken token, wtype *type, wvalue *value)*

The `wGetToken()` function retrieves data from WING, based on token `token`.

`wGetToken()` is a generic data retrieval call for `wapi`; Other retrieval functions described later in this document may be more tuned to your specific needs.
Depending on user actions, several data can queue in the receiving buffer; The function will use the data provided by the first occurrence of token `token` in the queue.
The data type returned by WING is used to set a more generic type in `type`, which can be one of int, float or string address on 32 bits data.
The actual value corresponding to int and float data is returned in `value`; In the case of string data (char*), either a NULL pointer is returned for no character present, or `value` contains the pointer to a string of characters.
Note that in the case a string is returned by `wGetToken()`, memory for storing the string will have been allocated by the function. It is the responsibility of the calling application to free the allocated memory when no longer needed to avoid application memory leaks.
`wGetToken()` will return `WSUCCESS` if the token token is found in the receiving queue and valid data is returned, `WZERO` if no valid data type is found or if a timeout occurs during receiving data.
`WMEMORY`, `WSENDERROR` or `WRECVERROR` can be returned in specific error cases.

Below is a small program example of using set() and get() calls, the receiving part using the `wGetToken()` function we first show the display obtained from running the program, followed by the program source code;

```
PS C:\Users\patri\eclipse\wtest\release> ./wtest
WING found at IP: 192.168.1.71
Using version 0.10
type = 3, data = RIDE
type = 2, data = -50.000000
type = 2, data = 0.000000
type = 1, data = 20
type = 2, data = 8.000000
type = 2, data = 0.486968
type = 2, data = 6.000000
```

```
#include <stdio.h>
#include <string.h>
//
#include "wapi.h"
#include "wext.h"
//
int main() {
        int             i;
        char            wingip[24] = "";
        wtype           type;
        wvalue          value;
        //
        if ((i = wOpen(wingip)) != WSUCCESS) exit(1);
        printf("WING found at IP: %s\n", wingip);
        printf("Using version %i.%i\n", wVer()/256, wVer()&15);
        //
        wSetTokenString(CH_1_GATE_MDL, "RIDE");       //Auto Rider Dynamics
        wSetTokenFloat(CH_1_GATE_1, -50.);            // thr
```

```
        wSetTokenFloat(CH_1_GATE_2, 0.);            // tgt
        wSetTokenInt(CH_1_GATE_3, 20);              // spd
        wSetTokenFloat(CH_1_GATE_4, 8.);            // ratio
        wSetTokenFloat(CH_1_GATE_5, 0.5);           // hold
        wSetTokenFloat(CH_1_GATE_6, 6.0);           // range
        //
        wGetToken(CH_1_GATE_MDL, &type, &value);
        if (value.sval) {
                printf("type = %i, data = %s\n", type, value.sval);
                free(value.sval);
        } else {
                printf("no data for ch 1 gate model!\n");
        }
        wGetToken(CH_1_GATE_1, &type, &value);
        printf("type = %i, data = %f\n", type, value.fval);
        wGetToken(CH_1_GATE_2, &type, &value);
        printf("type = %i, data = %f\n", type, value.fval);
        wGetToken(CH_1_GATE_3, &type, &value);
        printf("type = %i, data = %i\n", type, value.ival);
        wGetToken(CH_1_GATE_4, &type, &value);
        printf("type = %i, data = %f\n", type, value.fval);
        wGetToken(CH_1_GATE_5, &type, &value);
        printf("type = %i, data = %f\n", type, value.fval);
        wGetToken(CH_1_GATE_6, &type, &value);
        printf("type = %i, data = %f\n\n", type, value.fval);
        fflush(stdout);
        return(0);
}
```

int wGetTokenFloat(wtoken token, float fval)*

The `wGetTokenFloat()` function interrogates WING token `token` to get its currently associated value.
As it is the case for `wGetToken(), wGetTokenFloat()` will block until a token `token` is encountered in the receiving queue. The received token value has a given native type, and the function will do its best at converting the received data to float format as expected by the `fval` variable.
For example, inquiring WING token `CH_1_PEQ_1F` will return the current value of the token as a float value in `fval`. Inquiring WING token `CH_1_PEQ_ON` will result in a value of `0.0` or `1.0`, depending on the state of the token.
On the other hand, inquiring WING token `CH_1_NAME` will most likely return a value of `0.0` and a status of `WZERO`; In some cases (i.e., you set the name to be string **"123.7"** for example) you may get a valid floating-point value returned.
The function returns `WSUCCESS` if the requested operation was successful, other values can be returned, such as `WZERO` if no suitable format was found for adapting token value to `fval`, or `WSEND_TCP_ERROR` if an error took place while communicating with WING. Attempting to get a value from a token of type `NODE` will return `WNODE`.

int wGetTokenInt(wtoken token, int ival)*

The `wGetTokenInt()` function interrogates WING token `token` to get its currently associated value. As it is the case for `wGetToken(), wGetTokenInt()` will block until a token `token` is encountered in the receiving queue. The received token value has a given native type, and the function will do its best at converting the received data to integer format as expected by the `ival` variable.
For example, inquiring WING token `CH_1_PEQ_1F` will return the current value of the token as an int value in `ival`. Inquiring WING token `CH_1_PEQ_ON` will result in a value of `0` or `1`, depending on the state of the token.
On the other hand, inquiring WING token `CH_1_NAME` will return a value of `0` and a status of `WZERO`; In some cases (i.e., you set the name to be string **"12"** for example) you may get a valid int value returned.
The function returns `WSUCCESS` if the requested operation was successful, other values can be returned, such as `WZERO` if no suitable format was found for adapting token value to `ival`, or `WSEND_TCP_ERROR` if an error took place while communicating with WING. Attempting to get a value from a token of type `NODE` will return `WNODE`.

int wGetTokenString(wtoken token, char str)*

The `wGetTokenString()` function interrogates WING token `token` to get its currently associated value. As it is the case for `wGetToken(), wGetTokenString()` will block until a token `token` is encountered in the receiving queue. The received token value has a given native type, and the function will do its best at converting the received data to string/char* format as expected by the `str` variable.
For example, inquiring WING token `CH_1_PEQ_1F` will return the current value of the token as a string in `str`. Inquiring WING token `CH_1_PEQ_ON` will result in a 1-character string of "`0`" or "`1`", depending on the state of the token. Similarly, a token with a floating-point native format would result in a string containing the string representation of the floating-point value.
As a last example, inquiring WING token `CH_1_NAME` will return the string currently used for naming channel 1. Please note that the `str` variable should provide enough space to collect the data returned by `wGetTokenString()`.
The function returns `WSUCCESS` if the requested operation was successful, other values can be returned, such as `WZERO` if no suitable format was found for adapting token value to `str`, or `WSEND_TCP_ERROR` if an error took place while communicating with WING. Attempting to get a value from a token of type `NODE` will return `WNODE`.

*int wGetTokenDef(wtoken token, int *num, unsigned char* str)*

The `wGetTokenDef()` function interrogates WING token `token` to get its currently associated definition in raw form. As it is the case for `wGetToken(), wGetTokenDef()` will block until a token `token` is encountered in the receiving queue. The received token definition follows the description presented earlier in this document[39].
The returned data consists of the number of bytes `num` contained in an array of bytes `str`. Mote that `str` is allocated by the `wGetTokenDef()` function; it is therefore the responsibility of the calling application to free the allocated memory when no longer needed. Parsing this data is left to the application, and follows the description for node definition response.

The `Get()` functions presented above are all "one shot read" functions so to speak; They request data from WING, and wait for the right token to appear in the receiving queue. They will return the buffer content, adapting it to the requested type of data when applicable. This is a simple way to gather information from the console, but comes with a caveat if someone is also manipulating (locally or remotely) the desk. Indeed, as other changes take place and assuming your communication channel is in an 'open' state (i.e., your last communication with WING is less than 10s old), the console will natively send you changes that are taking place, resulting of the local or remote changes operated onto the desk.
So, when a "one shot read" request arrives and is served, it will sort through the received data for the expected token, and in doing this will discard the data received prior to finding the correct token.

wapi therefore provides another set of `Get()` functions for applications requiring a finer time control over the data they exchange with WING. In this new set of functions, the `Get()` instance will, as for the non-timed versions, gather information from WING and filter the possibly multiple[40] received tokens for the first one matching the specified token provided at call time, for a given amount of time only. Only when the specified token is received or time has expired (whichever comes first) will the function process the data it received, if available.

[39] See "Node Definition Response" in the "WING native / binary data interface" chapter
[40] Can literally be hundreds

*int wGetTokenTimed(wtoken token, wtype *type, wvalue *value, int timeout)*

`wGetTokenTimed()` is equivalent to its blocking sibling function call `wGetToken()`, but will block only for up to `timeout` microseconds; If no data corresponding to token `token` is received during that amount of time, the function will return `WZERO`. If a token `token` is received withing the time allocated by `timeout`, the function will parse data and return it as in the case of `wGetToken()`.

*int wGetTokenFloatTimed(wtoken token, float *fval, int timeout)*

The `wGetFloatTimed()` function is similar to the `wGetTokenFloat()` function in the sense it aims at retrieving from WING data and adapt it to floating-point format before returning it to `fval`;
But it will do so over a period `timeout`, expressed in µs (microseconds).
As long as timeout is not reached, the function is inquiring WING for data; If after timeout has expired, no data appears available, a value of `WZERO` is returned.
If on the contrary data is available from WING, the function will check if the data token is the correct one; it will treat the data as done in the `wGetTokenFloat()` function; i.e. the value retrieved from WING is converted to float format as expected by the `fval` variable.
For example, inquiring WING token `CH_1_PEQ_1F` will return the current value of the token as a float value in `fval`. Inquiring WING token `CH_1_PEQ_ON` will result in a value of `0.0` or `1.0`, depending on the state of the token.
On the other hand, inquiring WING token `CH_1_NAME` will most likely return a value of `0.0` and a status of `WZERO`; In some cases (i.e., you set the name to be string **"123.7"** for example) you may get a valid floating-point value returned.
The function returns `WSUCCESS` if the requested operation was successful, other values can be returned, such as `WZERO` if no suitable format was found for adapting token value to `fval`, or `WSEND_TCP_ERROR` if an error took place while communicating with WING. Attempting to get a value from a token of type `NODE` will return `WNODE`.
If data is available from WING and the data token is not the expected one, the function discards data and inquires WING for new data. The above takes place as long as timeout is not reached.

*int wGetTokenIntTimed(wtoken token, int *ival, int timeout)*

The `wGetTokenIntTimed()` function is similar to the `wGetTokenInt()` function in the sense it aims at retrieving from WING data and adapt it to floating-point format before returning it to `ival`;
But it will do so over a period `timeout`, expressed in µs (microseconds).
As long as timeout is not reached, the function is inquiring WING for data, if no data appears available, a value of `WZERO` is returned.
If on the contrary data is available from WING, the function will check if the data token is the correct one; it will treat the data as done in the `wGetTokenInt()` function; i.e. the value retrieved from WING is adapted to float format as expected by the `ival` variable.
For example, inquiring WING token `CH_1_PEQ_1F` will return the current value of the token as an int value in `ival`. Inquiring WING token `CH_1_PEQ_ON` will result in a value of `0` or `1`, depending on the state of the token.
On the other hand, inquiring WING token `CH_1_NAME` will return a value of `0` and a status of `WZERO`; In some cases (i.e., you set the name to be string **"12"** for example) you may get a valid int value returned.
The function returns `WSUCCESS` if the requested operation was successful, other values can be returned, such as `WZERO` if no suitable format was found for converting the retrieved value to `ival`, or `WSEND_TCP_ERROR` if an

error took place while communicating with WING. Attempting to get a value from a token of type `NODE` will return `WNODE`.
If data is available from WING and the data token is not the expected one, the function discards data and inquires WING for new data. The above takes place as long as timeout is not reached.

int wGetTokenStringTimed(wtoken token, char str, int timeout)*

The `wGetTokenStringTimed()` function is similar to the `wGetTokenString()` function in the sense it aims at retrieving from WING data and adapt it to string format before returning it to `str`;
But it will do so over a period `timeout`, expressed in µs (microseconds).
As long as timeout is not reached, the function is inquiring WING for data, if no data appears available, a value of `WZERO` is returned.
If on the contrary data is available from WING, the function will check if the data token is the correct one; `wGetTokenStringTimed()` will then treat the data as done in the `wGetTokenString()` function; The value retrieved from WING is adapted to string format as expected by the `str` variable. Similar restrictions and conversion rules apply. For example, inquiring WING token `CH_1_PEQ_1F` will return the current value of the token as a string in `str`. Inquiring WING token `CH_1_PEQ_ON` will result in a 1-character string of "`0`" or "`1`", depending on the state of the token. Similarly, a token with a floating-point native format would result in a string containing the string representation of the floating-point value. As a last example, inquiring WING token `CH_1_NAME` will return the string currently used for naming channel 1. Attempting to get a value from a token of type `NODE` will return `WNODE`.
If data is available from WING and the data token is not the expected one, the function discards data and inquires WING for new data. The above takes place as long as timeout is not reached.

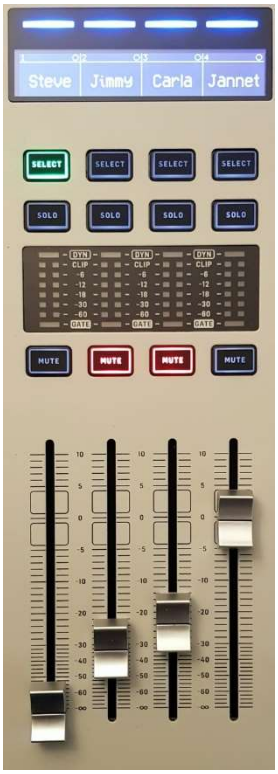
A Small Program Example

Let's program! Assume you need to programmatically change the name of channels and mute/unmute the respective channels from data contained in a file. Let's consider the file also contains initial channel faders, and covers channels 1 to 4. The file can be a text file such as:

```
Steve  0  -144.0
Jimmy  1  -30.0
Carla  1  -22.0
Jannet 0  -100.0
```

This is C source code; easy to understand and translate if needed to other programming languages.
We show on the right of the page the resulting channel strips 1-4:

```
#include <stdio.h>
#include <string.h>
//
#include "wapi.h"
#include "wext.h"
//
int main() {
    wtoken ntoken[] = {CH_1_NAME, CH_2_NAME, CH_3_NAME, CH_4_NAME};
    wtoken mtoken[] = {CH_1_MUTE, CH_2_MUTE, CH_3_MUTE, CH_4_MUTE};
    wtoken ftoken[] = {CH_1_FDR, CH_2_FDR, CH_3_FDR, CH_4_FDR};
    char   wingip[24] = "";    int    mute;
    float  fader;
    char   name[24];
    FILE*  fd;
    //
    // we don't know the IP of our console…
    if (wOpen(wingip)!= WSUCCESS) exit(1);
    printf("WING found at IP: %s\n", wingip);
    // open the file for reading
    if ((fd = fopen("file", "r")) != 0) {
        for (int i = 0; i< 4; i++) {
            // get data from the file
            fscanf(fd, "%23s %i %f", name, &mute, &fader);
            printf("%s %i %f\n", name, mute, fader);
            // set/send values to WING;
            // we don't care about the returned status
            wSetTokenString(ntoken[i], name);
            wSetTokenInt(mtoken[i], mute);
            wSetTokenFloat(ftoken[i], fader);
        }
    }
    fclose(fd);
    wClose();
    exit(0);
}
```



Event-driven updates

There are times and situations when WING will send data to your program. This has been explained above: As soon as you are connected to WING and have exchanged data with it, the connection will stay in an open state for 10s, unless you specifically establish and close the TCP connection around your work. While this will help, it will not prevent WING to send you data while the TCP link is active, and is certainly not an effective way to manage data as you will send more resources in opening/closing the connection than in time sending or receiving data.

wapi provides additional API functions to manage event driven applications. These are managed around the notion of 'main loop' as often found in IOT devices running Arduino devices, in standard Linux or Windows applications where a main loop ensures the management of all events coming from devices connected to your computer (mouse, keyboard, etc.). WING data can be treated just as any other event.

API calls are therefore available to keep a connection between your application and WING alive, as well as to get data from WING, effectively emptying the event queue of the communication with the console. This will be assured with the `wKeepAlive()` and the `wGetVoidPTokenTimed()` function calls presented below.

int wKeepAlive

`wKeepAlive()` maintains the connection between WING and the calling program so data issued by the console with no request initiated by the program can be received in a main loop, or over an extended period of time beyond 10s[41].

In fact, this function can be called as often as you like and will optionally performs a small exchange with the console, based on an internal timer. The elapsed time between two effective exchanges of data with the console depend on the value of `wKeepAlive_TIMEOUT` which is part of the `wapi.h` file.

The `wKeepAlive()` function returns `WSUCCESS` if a valid exchange took place to renew a 10 seconds working communication, or `WZERO` if no exchange was necessary. The function can also return the values of `WSEND_ERROR` or `WRECV_ERROR` if communication was not successful.

*int wGetParsedEvents(wTV *tv, int maxevents)*

The `wGetParsedEvents()` API call is a specific `Get` function. Unlike other `Get` functions previously presented in this document, it does not expect data from a specific token, nor a specified format in which the data from the console should be converted to. The function will check the WING receive event queue for data and will only return when data is received by removing the events available from the oldest queue record. If no valid data is found, the function eventually returns with a network error.

When data is available in the event queue events are retrieved in a FIFO order, and the token, type and data associated to events are returned to the calling application using the `tv` structure array. The function returns the number of events parsed if data has been returned to the calling program, `WZERO` if no events were found. It can also return `WMEMORY` on memory allocation errors or `WRECV_ERROR` on TCP read errors. The parameter `maxevents` represents the maximum number of entries `tv` can accept; The function will allocate memory for its event read buffer to match that size.

*int wGetParsedEventsTimed(wTV *tv, int maxevents, int timeout)*

The `wGetParsedEventsTimed()` API call is similar to the `wGetParsedEvents()` function, but will returned after a maximum time of `timeout` in µs. Like in the case of the `wGetParsedEvents()` function, `wGetParsedEventsTimed`()

[41] The actual value in the wapi library may vary (but less than 10s) to ensure stability in event driven communications

does not expect data from a specific token, nor a specified format in which the data from the console should be converted to. The function will check the WING receive event queue for data and will only return when data is received by removing the events available from the oldest queue record. If no data is found before a timeout of `timeout` µs, the function returns with a value `WZERO`.
If data is available in the event queue, events are retrieved in a FIFO order, and the token, type and data associated to events are returned to the calling application using the `tv` structure array. The function returns the number of events parsed if data has been returned to the calling program, `WZERO` if no events were found. It can also return `WMEMORY` on memory allocation errors or `WRECV_ERROR` on TCP read errors. The parameter `maxevents` represents the maximum number of entries `tv` can accept; The function will allocate memory for its event read buffer to match that size.

In a typical, simple example of use of the two API calls shown in the following paragraph, the main loop is replaced with a `while(1){}` statement.

```
#include <stdio.h>
#include <string.h>
//
#include "wapi.h"
#include "wext.h"
//
int main() {
    int   i, j;
    char  wingip[24] = "";
    wTV   TV[100];

    if ((i = wOpen(wingip)) != WSUCCESS) return(-1);
    printf("WING found at IP: %s\n", wingip);
    printf("Using version %i.%i\n", wVer()/256, wVer()&15);

    while (1) {
        wKeepAlive();
        //
        if ((i = wGetParsedEventsTimed(TV, 100, 1000)) > 0) {
             for (j = 0; j < i; j++) {
                printf("W-> %s type = %i, data = ", wGetName(TV[j].token), TV[j].type);
                if (TV[j].type == I32) printf("%i\n", TV[j].d.idata);
                if (TV[j].type == F32) printf("%.2f\n", TV[j].d.fdata);
                if (TV[j].type == S32) {
                    printf("%s\n", TV[j].d.sdata);
                    if (TV[j].d.sdata) free(TV[j].d.sdata);
                }
                fflush(stdout);
             }
        } else {
            if (i != WZERO) {
                printf("Error = %i\n", i); fflush(stdout);
            }
        }
    }
    return 0;
}
```

An example of (partial) output of the code snippet above, after launching the program and manipulating ch.1, 2 and 3 mutes and ch. 1 fader:

```
PS C:\Users\patri\eclipse\wtest\release> ./wtest
WING found at IP: 192.168.1.71
Using version 0.10
W-> ch.1.mute type = 1, data = 1
W-> ch.2.mute type = 1, data = 1
W-> ch.3.mute type = 1, data = 1
W-> ch.1.fdr type = 2, data = -144.00
W-> ch.1.fdr type = 2, data = -89.18
W-> ch.1.fdr type = 2, data = -87.07
W-> ch.1.fdr type = 2, data = -85.55
W-> ch.1.fdr type = 2, data = -84.49
```

Nodes

In many applications as well as in browsing over the `JSON` data structure, one can easily envision it would be interesting for optimization purposes to get and set a group of attributes at once, rather than establishing communication requests for each single parameter.

Nodes were introduced in the X32 family to enable this functionality, and have been widely used in several applications for controlling the desk; In the case of WING this may be even more interesting due to the very large number/volume of parameter data available as one unrolls each branch in the `JSON` tree opening a new level of nodes and parameters. Each branch of the `JSON` tree can be walked through by a program, resulting in a (sometimes very large) set of {`token`, `value`} sets and a way to represent the depth in the hierarchical tree the reported sets are issues from.

We show below the node data extracted (using a `wapi` call) for a few nodes:

```
wing_root: {$stat{}, cfg{}, $syscfg{}, io{}, ch{}, aux{}, bus{}, main{}, mtx{}, dca{}, fx{},
cards{}, play{}, rec{}}

node $stat, size: 212,
.modtype=NONE,A.stat=-,dev="",.B.stat=-,dev="",.C.stat=-,dev="",.lock=1,ppm=0,solo=0,sip=0,rtcerr=
0,time=19:14:44,date=2020-12-08,usbstate=IDLE,usbvolname=INTENSO,sc_stat=-,sc_devices="",sc_upcnt=
32,sc_dncnt=32,

node $syscfg, size: 187,
.consolename=HMS01,logflags="",ipmode=STATIC,ip0=192,ip1=168,ip2=1,ip3=71,msk0=255,msk1=255,msk2=2
55,msk3=0,gw0=192,gw1=168,gw2=1,gw3=254,$ipapply=0,$firmware=1.0922g8a2cd9b2:develop,
```

As mentioned above, some nodes such as `ch`, are very large (more than 80k characters).

The set of values in a node list can be variable depending on the options (effects for example) loaded in the console at the time of the call, but all tokens are fixed and only contain known data types; A node can be set and retrieved as a single line of text with pre-formatted data, making it easy to store and manage in applications. `wapi` offers two methods of saving node data sets from the console, the first one is returning data similar to what is obtained using `OSC` (text data); the other one is more suitable to direct use from a compiled program, with binary data saved in specific structures containing the token and its respective data. The first method typically takes 2 or 3 seconds to get all WING data as multiple strings of node data. The second method is probably more suitable for use with `wapi` and is also faster (1 to 2 seconds) as no formatting is involved in saving the data returned by the console.

The following functions list API entries to use WING nodes as defined above.

*int wSetNode(char *str)*

The `wSetNode()` function parses the string contained in `str` according to the format used in OSC nodes; For example, a string such as `/ch.1.fdr 8.5,mute 1,/bus.1.fdr 5.0,.2.fdr=0.5` will set fader of channel 1 to the 8.5dB value and mute the channel. Bus 1 fader will be set to 5dB and bus 2 fader will be set to 0.5dB.
Each parameter group is separated by a '`,`' character, the '`/`' character represents the root of the `JSON` parameter tree, and '`.`' characters are used to navigate up and down within the `JSON` parameter tree.
The function returns a status `WSUCCESS` if the string was processed with no errors; It will return `WNODE` if a token or value provided with the string `str` is not valid. The function can also report other errors if communication issues were detected. str must be `\0` ended. Please see an example of use below.

*int wSetNodeFtomTVArray(wTV *array, int nTV)*

The `wSetNodeFromTVArray()` function sends updates to `WING` in a single network exchange from the `nTV` elements in `wTV` (see below) array `array`; This is a great way to improve network performance. Although the function is the symmetrical to `wGetNodeToTVArray()`, it can accept hierarchically organized elements or uncorrelated elements as long as they are not nodes and contain valid tokens-values sets. The function returns `WSUCCESS` or an error if one takes place during allocating, preparing, or sending the resulting network buffer to `WING`.

*int wSetBinaryNode (unsigned char *array, int len)*

The `wSetBinaryNode()` function will load from `array` the raw, the `len` bytes of binary data commands to be be executed by the desk in a single call. The function returns `WSUCCESS` or an error if one takes place during sending the buffer to `WING`.
A typical use for this call is external scene management[42] such as in the code snippet below.

```
void WRestoreS() {
        unsigned char node[256000];
        int i, j;
        //
        if ((fd = fopen("Scene.scn", "rb")) != NULL) {
                fread(&i, sizeof(int), 1, fd);
                fread(node, i * sizeof(unsigned char), 1, fd);
                if ((j = wSetBinaryNode(node, i)) < i) {
                        printf("Error: %d only bytes sent vs. %d\n", j, i);
                } else {
                        printf("Restored Scene\n");
                }
                fclose(fd);
        } else {
                printf("Cannot open Scene.scn\n");
        }
        Return;
}
```

[42] As opposed to WING internal Show files and Scene entities (see dedicated chapter)

*int wGetNode(wtoken node, char *str)*

The `wGetNode()` function will return in `str` a string of values separated formatted as in the `OSC` node convention and corresponding to the node token `node`.
`str` must be large enough to accept the characters returned by the call. The function returns a status `WSUCCESS` if the node was processed with no errors; It will return `WTOKEN` if the token provided is not a valid node and `WNODE` if an error occurs during parsing the data received from the console. The function can also report other errors if communication issues were detected. The line of text returned by the function end with a line-feed and a `\0` byte.

*int wGetNodeToTVArray (wtoken node, wTV *array)*

The `wGetNodeToTVArray()` function will return in TV, an array of structures `wTV` (see below), all values respective of their corresponding token and part of the node token `node`.
`array` must be large enough to accept the data returned by the call (see below for the number of elements for each level-1 node). The function the number of tokens in the array `array` if the node was processed with no errors; It will return `WTOKEN` if the token provided is not a valid node and `WNODE` if an error occurs during parsing the data received from the console. The function can also report other errors if communication issues were detected.

Below is an indicative value of the number of `wTV` structures in the returned arrays for each level-1 node of the console at the time of this writing; The sum of the values following 'size:' give an idea of the number of parameters the console knows.

```
node $stat, size: 20
node cfg, size: 156
node $syscfg, size: 17
node io, size: 5699
node ch, size: 10216
node aux, size: 1460
node bus, size: 2208
node main, size: 424
node mtx, size: 736
node dca, size: 144
node mgrp, size: 16
node fx, size: 96
node cards, size: 71
node play, size: 118
node rec, size: 7
node $ctl, size: 2357
```

Below is a C code source example of use of `WSetNode()`; The program will set the faders of channels 1 to 4 to different positions, unmute channels 1 and 3, channel 2 mute is unchanged and channel 4 will be muted. DCA 1 is muted and its fader is set to 1dB.

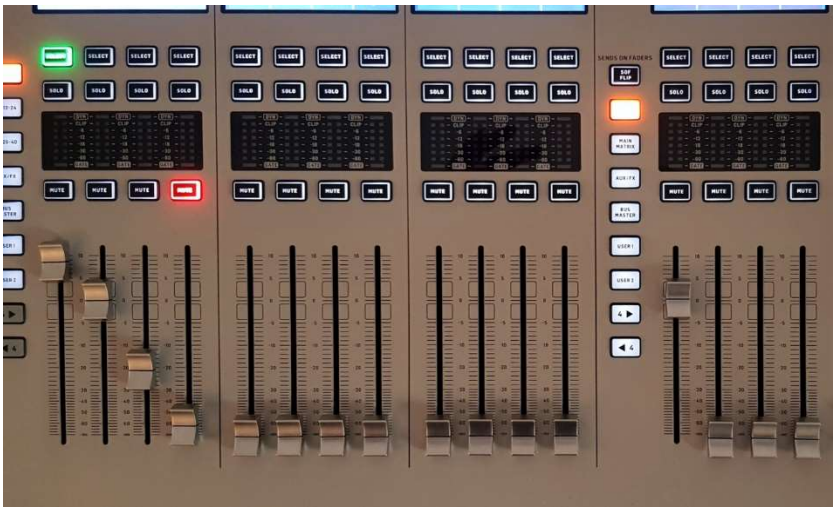
```
/*
 * wtest.c
 *
 *  Created on: Oct. 18, 2020
 *      Author: Patrick-Gilles Maillot
 */
#define _WIN32_WINNT  0x501
//
#include <stdio.h>
#include <string.h>
//
```

```
#include "wapi.h"
#include "wext.h"
//
int main() {
        int     i;
        char    wingip[24] = "";

        char    wtest1[64] = "/ch.1.fdr 8.5,mute=0,.2.fdr=0,/dca.1.fdr=1.0,mute=0";
        char    wtest2[64] = "/ch.3.fdr -20,mute=0,.4.fdr=-60,mute=1";
        //
        if ((i = wOpen(wingip)) != WSUCCESS) exit(1);
        printf("WING found at IP: %s\n", wingip);
        printf("Using version %i.%i\n", wVer()/256, wVer()&255);
        //
        i = wSetNode(wtest1);
        printf("result = %d, initial data: %s\n", i, wtest1);
        i = wSetNode(wtest2);
        printf("result = %d, initial data: %s\n", i, wtest2);
        fflush(stdout);
        return(0);
}
```

A listing of the program when ran:

```
PS C:\Users\patri> cd eclipse/wtest/release
PS C:\Users\patri\eclipse\wtest\release> ./wtest
WING found at IP: 192.168.1.71
Using version 0.6
result = 1, initial data: /ch.1.fdr 8.5,mute=0,.2.fdr=0,/dca.1.fdr=1.0,mute=0
result = 1, initial data: /ch.3.fdr -20,mute=0,.4.fdr=-60,mute=1
PS C:\Users\patri\eclipse\wtest\release>
```

Below the resulting state of the console, starting from an init state:



Requesting the full set of nodes from a freshly initialized console[43] results in a file of 200000+ characters, and is therefore a lot of data to manage. Over WIFI, it takes about 2 seconds to execute a full dump as OSC-like node data and 1 to 2 seconds to retrieve a full dump as `wTV` structures.

We show below a typical example of the `OSC`-like node string for `ch.1` returned by **wapi** when using `wGetNode`():

[43] FW 1.10

```
node ch:
.1.in.set.$mode=M,srcauto=0,altsrc=0,inv=0,trim=0.00,bal=0.00,$g=0.00,$vph=0,dly=0.00,.conn.grp=LC
L,in=1,altgrp=OFF,altin=1,..flt.lc=0,lcf=100.24,hc=0,hcf=10018.26,tf=0,mdl=TILT,tilt=0.00,.clink=0
,col=1,name="",icon=1,led=1,mute=0,fdr=144.00,pan=0.00,wid=100.00,$solo=0,$sololed=0,solosafe=0,mo
n=A,proc=GEDI,ptap=4,$presolo=0,peq.on=0,1g=0.00,1f=99.69,1q=2.00,2g=0.00,2f=999.25,2q=2.00,3g=0.0
0,3f=10016.53,3q=2.00,.gate.on=0,mdl=GATE,thr=40.00,range=40.00,att=10.00,hld=10.00,rel=199.40,acc
=0.00,ratio=1:3,.gatesc.type=OFF,f=1002.37,q=2.00,src=SELF,tap=IN,$solo=0,.eq.on=0,mdl=STD,mix=100
.00,$solo=0,$solobd=1,lg=0.00,lf=80.20,lq=2.00,leq=SHV,1g=0.00,1f=200.00,1q=2.00,2g=0.00,2f=601.39
,2q=2.00,3g=0.00,3f=1499.79,3q=2.00,4g=0.
00,4f=3990.52,4q=2.00,hg=0.00,hf=11994.42,hq=2.00,heq=SHV,.dyn.on=0,mdl=COMP,mix=100.00,gain=0.00,
thr=10.00,ratio=3.00,knee=3,det=RMS,att=50.00,hld=20.00,rel=152.57,env=LOG,auto=1,.dynxo.depth=6.0
0,type=OFF,f=1002.37,$solo=0,.dynsc.type=OFF,f=1002.37,q=2.00,src=SELF,tap=IN,$solo=0,.preins.on=0
,ins=NONE,$stat=,.main.1.on=1,lvl=0.00,.2.on=0,lvl=0.00,.3.on=0,lvl=0.00,.4.on=0,lvl=0.00,..send.1
.on=0,lvl=144.00,pon=0,ind=0,mode=PRE,plink=0,pan=0.00,wid=100.00,.2.on=0,lvl=144.00,pon=0,ind=0,m
ode=PRE,plink=0,pan=0.00,wid=100.00,.3.on=0,lvl=144.00,pon=0,ind=0,mode=PRE,plink=0,pan=0.00,wid=1
00.00,.4.on=0,lvl=144.00,pon=0,ind=0,mode=PRE,plink=0,pan=0.00,wid=100.00,.5.on=0,lvl=144.00,pon=0
,ind=0,mode=PRE,plink=0,pan=0.00,wid=100.00,.6.on=0,lvl=144.00,pon=0,ind=0,mode=PRE,plink=0,pan=0.
00,wid=100.00,.7.on=0,lvl=144.00,pon=0,ind=0,mode=PRE,plink=0,pan=0.00,wid=100.00,.8.on=0,lvl=144.
00,pon=0,ind=0,mode=PRE,plink=0,pan=0.00,wid=100.00,.9.on=0,lvl=144.00,pon=0,ind=0,mode=PRE,plink=
0,pan=0.00,wid=100.00,.10.on=0,lvl=144.00,pon=0,ind=0,mode=PRE,plink=0,pan=0.00,wid=100.00,.11.on=
0,lvl=144.00,pon=0,ind=0,mode=PRE,plink=0,pan=0.00,wid=100.00,.12.on=0,lvl=144.00,pon=0,ind=0,mode
=PRE,plink=0,pan=0.00,wid=100.00,.13.on=0,lvl=144.00,pon=0,ind=0,mode=PRE,plink=0,pan=0.00,wid=100
.00,.14.on=0,lvl=144.00,pon=0,ind=0,mode=PRE,plink=0,pan=0.00,wid=100.00,.15.on=0,lvl=144.00,pon=0
,ind=0,mode=PRE,plink=0,pan=0.00,wid=100.00,.16.on=0,lvl=144.00,pon=0,ind=0,mode=PRE,plink=0,pan=0
.00,wid=100.00,..postins.on=0,mode=FX,ins=NONE,w=0.00,$stat=,.tags="",$fdr=144.00,$mute=0,$muteovr
=0,
```

*int wGetBinaryNode (wtoken node, unsigned char *array, int maxlen)*

The `wGetBinaryNode()` function will return in `array` the raw, binary data corresponding to the `WING` node selected with token `node`. The storage buffer `array` must be large enough to accept the data returned by the call, up to `maxlen` bytes. The function returns the number of bytes saved in `array`, or an error status if less or equal to 0.

A typical use for this call is external scene management[44] such as in the code snippet below.

```
void WSaveS() {
        unsigned char node[2048];
        int i;
        //
        if ((fd = fopen(Scene.scn, "wb")) != NULL) {
                // Get the node data for CH_1, composed of many 0xd7<Wing data> parts
                if ((i = wGetBinaryNode(CH_1, node, 2048)) < WSUCCESS) {
                        fclose(fd);
                        printf("Error reading node %d\n", tarray[ta]);
                        return;
                }
                fwrite(&i, sizeof(int), 1, fd);
                fwrite(node, i * sizeof(unsigned char), 1, fd);
                printf("Saved scene\n");
                fclose(fd);
        } else {
                printf("Cannot create Scene.scn\n");
        }
        return;
}
```

[44] As opposed to WING internal Show files and Scene entities (see dedicated chapter)

*int wGetBinaryData (char *str, unsigned char *array, int maxlen)*

The `wGetBinaryData()` function will return in `array` the raw, binary data corresponding to the `WING` node or parameter represented by its text description `str`. The storage buffer `array` must be large enough to accept the data returned by the call, up to `maxlen` bytes.

`str` is a string of characters enabling to select a single WING node or parameter, for example:

"/ch" (a node),
"/ch.1.eq" (a node), or
"/ch.1.name" (a parameter).

The function returns the number of bytes saved in `array`, or an error status if less or equal to 0.

Meters

WING offers many measurement points along the digital audio path; As a result, there are numerous meters. As briefly presented in a table earlier in this document, every Channels, Aux, Bus, Main and Matrix strip offers no less than 8 meters: input left & right, output left & right, gate & dyn key & gain. This alone represents 608 meters that can be retrieved, and there are much more.
The network data path for getting meter values is separated from the main network communication in order to keep things simpler for the programmer and sound engineer.
Meter data is transmitted to a `UDP` port chosen by the user. When selecting which meters to receive, the user associates an ID to the request, enabling simpler identification of the received data. As soon as a valid meter request is received, WING will send back the respective meter data for 5 seconds, every approximately every 50ms. In order to continue or continuously receive a set of meter data, the user must renew the request for data by issuing a simple renew command containing the ID of the requested meter set.

Meters API

wapi offers a small set of function calls to help programmers manage meter data. it hides the networking complexity and proposes a simple way of selecting what meter to get back from the digital console. Meter data will be provided back to the application in the form of a buffer of values, decoding data being left to the application.

int wMeterUDPPort (int wport)

The `wMeterUDPPort()` API call enable users to select the UDP port WING will send meter data to. It also prepares the `wapi` internal network for receiving meter data and being able to return data to the user application. `wport` is a standard UDP port and must be available for receiving data. The function returns WSUCCESS if everything is set correctly or will return an error value if the request was not successful.

*int wSetMetersRequest(int reqID, unsigned char *wMid)*

`wSetMetersRequest()` must be called in order to start receiving meter data. The API associates a request ID `reqID` to a selection of meters to receive. The request ID helps renewing the request for data and sorting through potentially multiple data sets sent by the console. The `wMid` parameter holds the selection of meters that can be recovered from WING in an array of `29` bytes. Each bit (from left to right) in the array of `29` bytes represents a meter family that can be received from the console, and is shown in the table below:

byte index	bits	selection
0-4	1-40	Channel 1-40
5	1-8	Aux 1-8
6-7	1-16	Bus 1-16
8	1-4	Main 1-4
9	1-8	Matrix 1-8
10	1-8	DCA 1-8
11-12	1-16	FX proc 1-16
13-14	1-16	Source input 1-16
15-16	1-11	Output 1-11
17	1	Monitor
18	1	RTA
19-23	1-40	Channel V2 1-40
24	1-8	Aux V2 1-8
25-26	1-16	Bus V2 1-16
27	1-4	Main V2 1-4
28	1-8	Matrix V2 1-8

For example, a C source language array declaration as follows will request meters for channels 1 and 40:

```
unsigned char   mbits[29] = {0x80, 0, 0, 0, 0x01, 0};    // bytes indexes 5 to 28 are 0
```

int wRenewMeters(int reqID)

The `wRenewMeters()` API call is used to renew a previous request for meter data; This function should be called every 5 seconds maximum in order to avoid losing meter data if continuous receiving is expected. The `reqID` parameter must be previously defined with a call to `wSetMetersRequest()`. The function returns `WSUCCESS` if the request is accepted, or will return other error status values otherwise.

*int wGetMeters(unsigned char *buf, int maxlen, int timeout)*

`wGetMeters()` will check if meter data has been received or is available. The call can be blocking or un-blocking depending on the value of `timeout`. A `timeout` of `0` will block the application in reading mode until data is available. A non-zero value of `timeout`, expressed in micro-seconds will return after the provided value and return to the caller with a value of `WZERO (0)` if no data is available or will return sooner with the actual number of bytes read available in `buf.`
The `maxlen` parameter indicates the maximum number of bytes `buf` can hold. It is the responsibility of the application to ensure `buf` is large enough to accept `maxlen` bytes.

The data returned by the wGetMeters() function is coded as follows:

```
<reqID><[meter data group][meter data group]  … >
```

Each `meter data group` is composed of several big-endian 16bits integers typically representing meter values expressed in `1/256`th of a dB, or otherwise returned data (for ex. `Gate led` returns `0` or `1`).

The table below provides the number and origin of each meter data for each of the possible meter groups:

Group name	Contents
channel aux bus main matrix	input left input right output left output right gate key gate gain dyn key dyn gain
dca	pre fader left pre fader right post fader left post fader right
fx	input left input right output left output right state meters (1…6)
source	source group levels (i.e. local ins: 8 meters)
output	output group levels (i.e. local outs: 8 meters)
monitor	solo bus left solo bus right mon 1 left mon 1 right mon 2 left mon 2 right
rta	rta slot meters (120)
channel V2 aux V2	input left input right

bus V2 main V2 matrix V2	output left output right gate key gate gain gate led dyn key dyn gain dyn state automix gain

Below is an example of buffers received after requesting meter data for channel 1 and using different sources, with `Ch 1` fader set to `+3dB`.
As received data uses signed `16bits` (`[-32768…+32767]`) and is expressed in `1/256`th of dB, the actual meter value can be calculated as `<int16>/256`.
Note that fx return a different format for their meters, with `value` in dB `= return value * 6.0 / 2048`.

```
                                    gate gate  dyn  dyn
          <reqID>  inL  inR outL outR  key gain  key gain

W→ 20 B: 00000002 9bb2 9bb2 8000 8000 9b7c 0000 8000 0000 (no input)
                  -100 -100 -128 -128 -100    0 -128    0
W→ 20 B: 00000002 f9fb f9fb fcfb fcfb f9fb 0000 ee01 0000 (OSC 1kHz, -6dB)
                    -6   -6   -3   -3   -6    0  -17    0
W→ 20 B: 00000002 d7fd d7fd dafd dafd d7fd 0000 aa02 0000 (OSC 1kHz, -40dB)
                   -40  -40  -37  -37  -40    0  -85    0
```

RTA test program

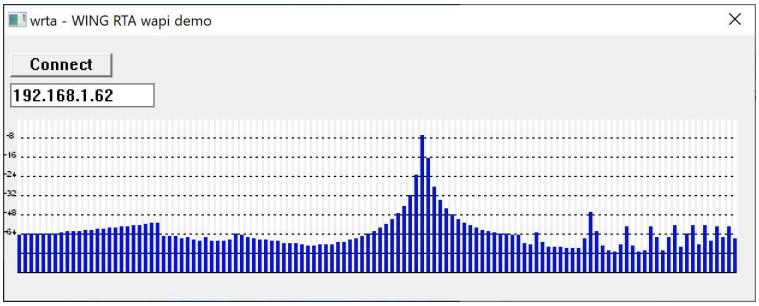
We show here is a small C / Windows program example showing how to get and display RTA. The scaling of meter data is tweaked in order to provide better readability, but isn't meant to be dB accurate.

```
/*
 * wrta.c
 *
 *  Created on: May 9, 2020
 *      Author: Patrick-Gilles Maillot
 *
 */
#include <windows.h>
#include <stdio.h>
#include <sys/time.h>
#include "wapi.h"
#include "wext.h"

// Windows Declarations
WINBASEAPI HWND WINAPI GetConsoleWindow(VOID);
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
//
HINSTANCE         hInstance = 0;
HWND              hwndipaddr, hwndconx;
HDC               hdc;
PAINTSTRUCT       ps;
MSG               wMsg;
HFONT             hfont;
HPEN              wnpen;                    // no line
HBRUSH            bBrush, rBrush, wBrush;   // blue, red, white
int               keep_running = 1;         // mainloop control
int               ready = 0;                // ready flag after connect OK
char              wingip[24] = "";          // let wapi tell us our IP
int               M_id = 3;                 // meters request ID
int               M_port = 10026;           // meters UDP port

#define MAXLEN    254                       // enough for RTA (244 bytes)
unsigned char     buf[MAXLEN];              // data buffer
int                len;

time_t            before = 0;               // timers
time_t            now;

unsigned char    mbits[29] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                              0, 0, 0, 0, 0, 0, 0, 0, 0x80,}; // RTA
                                           // remaining bytes/bits to 0


//
// void wRTAMeters()
// A basic RTA; height is 128, width is 5 per freq. (120 Freqs)
void wRTAMeters(int basex, int basey, int value) {
    int basexx = basex + 5;
    int baseyy = basey + 128;
    basex++;
    // not trying to be accurate, but close to WING data behavior
    //
    SelectObject(hdc, bBrush);
    SelectObject(hdc, wnpen);
    Rectangle(hdc, basex, baseyy, basexx, baseyy - min(value, 128));
    if (value > 120) {
        SelectObject(hdc, rBrush);
    } else {
        SelectObject(hdc, wBrush);
    }
    Rectangle(hdc, basex, baseyy - value, basexx, basey);
}
//
// Windows main function and main loop
//
int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PWSTR lpCmdLine, int nCmdFile) {
//
    union {
        unsigned char    cc[2];
        short            ii;
```

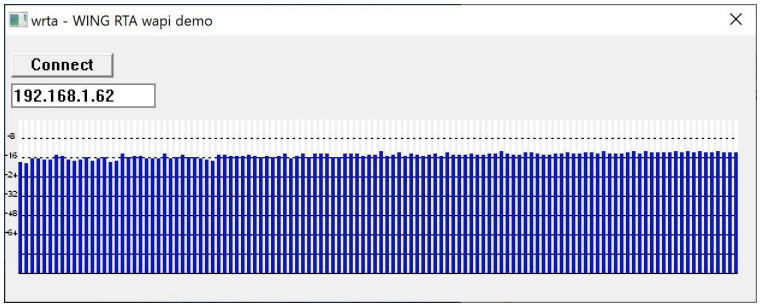
```
    } endian;
    int                 ival = 0;
    float               fval;
    //
    WNDCLASSW wc = {0};
    wc.lpszClassName = L"WING RTA";
    wc.hInstance = hInstance;
    wc.hbrBackground = GetSysColorBrush(COLOR_3DFACE);
    wc.lpfnWndProc = WndProc;
    wc.hCursor = LoadCursor(0, IDC_ARROW);
    //
    RegisterClassW(&wc);
    CreateWindowW(wc.lpszClassName, L"wrta - WING RTA wapi demo",
        WS_OVERLAPPED | WS_VISIBLE | WS_SYSMENU,
        220, 220, 630, 250, 0, 0, hInstance, 0);
    //
    // Main loop
    while (keep_running) {
        if (PeekMessage(&wMsg, NULL, 0, 0, PM_REMOVE)) {
            TranslateMessage(&wMsg);
            DispatchMessage(&wMsg);
        }
        if (ready) {
            now = time(NULL);                   // maintain meters
            if (now > before + 4) {             // by sending
                wRenewMeters(M_id);             // request every less than
                before = now;                   // 5 seconds
            }
            // Read meters (if any data) with a timeout of 10ms
            if ((len = wGetMeters(buf, MAXLEN, 10000)) > 0) {
                for (int i = 0; i < 120; i++) {
                    endian.cc[0] = buf[5 + i + i];    // get data as short
                    endian.cc[1] = buf[4 + i + i];    // int
                    fval = ((float)(endian.ii + 32768)/32768.);
                    ival = fval * fval * 128;         // approximate conversion
                    wRTAMeters(10 + i * 5, 70, ival);
                }
            }
        }
    }
    return (int) wMsg.wParam;
}
//
//
LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam) {
    //
    char *str1[] = {" -8", "-16", "-24", "-32", "-48", "-64", "", ""};

    switch (msg) {
    case WM_CREATE:
        hwndipaddr = CreateWindow("Edit", NULL, WS_CHILD | WS_VISIBLE | WS_BORDER,
                5, 40, 120, 20, hwnd, (HMENU)0, NULL, NULL);
        hwndconx = CreateWindow("button", "Connect", WS_VISIBLE | WS_CHILD,
                5, 15, 85, 20, hwnd, (HMENU)1, NULL, NULL);
        break;
    case WM_PAINT:
        hdc = BeginPaint(hwnd, &ps);
        SelectObject(hdc, hfont);
        SetBkMode(hdc, TRANSPARENT);
        for (int i = 0; i < 8; i++) {
            TextOut(hdc, 0, 70 + i*16+10, str1[i], strlen(str1[i]));
            MoveToEx(hdc, 11, 70 + i*16+15, NULL);
            LineTo(hdc, 609, 70 + i*16+15);
        }
        break;
    case WM_COMMAND:
        if (HIWORD(wParam) == BN_CLICKED) {    // user action
            switch (LOWORD(wParam)) {
            case 1:
                if (ready) {
                    keep_running = 0;
                    PostQuitMessage(0);
                } else {
                    // Connect clicked
                    if (wOpen(wingip) != WSUCCESS) exit(1);
                    SetWindowText(hwndipaddr, wingip);
                    // set port to receive UDP data and request meters for RTA
```

```
                    if (wMeterUDPPort(M_port) != WSUCCESS) exit(1);
                    if (wSetMetersRequest(M_id, mbits) != WSUCCESS) exit(1);
                    ready = 1;
                }
                break;
            }
        }
        break;
    case WM_DESTROY:
        keep_running = 0;
        PostQuitMessage(0);
        break;
    }
    return DefWindowProcW(hwnd, msg, wParam, lParam);
}
//
//
// main program
int main(int argc, char **argv) {
    HINSTANCE hPrevInstance = 0;
    PWSTR pCmdLine = 0;
    int nCmdFile = 0;
    // Hide console window
    ShowWindow(GetConsoleWindow(), SW_HIDE);
    // Set colors, pens, fonts used in the app
    bBrush = CreateSolidBrush(RGB(10, 20, 200));
    rBrush = CreateSolidBrush(RGB(255, 30, 30));
    wBrush = CreateSolidBrush(RGB(255, 255, 255));
    wnpen = CreatePen(PS_NULL, 0, RGB(0, 0, 0));
    hfont = CreateFont(8, 0, 0, 0, FW_MEDIUM, 0, 0, 0,
            DEFAULT_CHARSET, OUT_OUTLINE_PRECIS, CLIP_DEFAULT_PRECIS,
            ANTIALIASED_QUALITY, VARIABLE_PITCH, TEXT("Arial"));
    // Launch program
    wWinMain(hInstance, hPrevInstance, pCmdLine, nCmdFile);
    wClose();
    return 0;
}
```

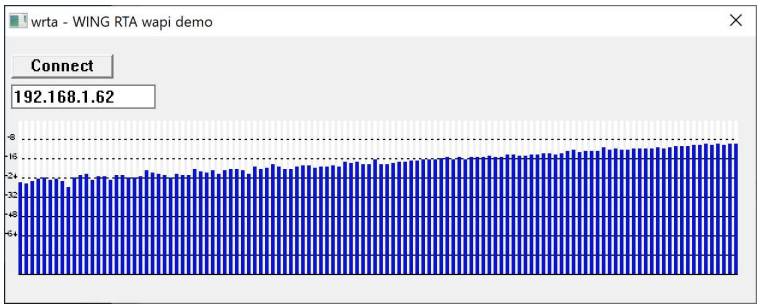
And the resulting displays: Sine wave signal 1kHz:



Pink noise mode:



White noise mode:



Effects and Plugins

WING comes with an impressive number of effects, plugins and emulations that can be used on any channel without costing any FX slots. In every channel, Gate, EQ Compressor can take different processing models you can organize and change on the fly. The following pages below present the different effects and their parameters.

Plugins

Plugins entries are directly included with channels, busses, etc. and can either default to WING standard algorithms or adapt to alternative plugins to color your sound or fit your taste when it comes to mixing. Plugins are showing under the main `JSON` structure, only when instantiated. WING Channel audio engines enable 4 sorts of plugins: Filter, Gate, EQ and Dynamics. Bus, Main and Matrix audio engines support EQ and Dynamics plugins.

The choice of plugin is represented by the name (or model) of the plugin, as set under the respective "`mdl`" token; After a console reset, the default channel Filter, Gate, EQ and Dynamics plugins will be "`TILT`", "`GATE`", "`STD`", and "`COMP`", respectively, and these can be changed to one of the multiple plugins available within the console (respecting the category they apply to of course).

The choice of plugin is represented by the name (or model) of the plugin, as set under the respective "`mdl`" token; authorized values are:

Filters:

```
TILT EQ, MAXER, AP 90, AP 180
```

Gates:

```
GATE/EXPANDER, DUCKER, EVEN 88 GATE, SOUL 9000 GATE, DRAW MORE 241, BDX902 DEESSER, WAVE
DESIGNER, DYNAMIC EQ, SOUL WARMTH PRE, 76 LIMITER AMP, LA LEVELER, AUTO RIDER, SOURCE
EXTRACTOR
```

Equalizers:

```
WING EQ, SOUL ANALOGUE, EVEN 88 FORMANT, EVEN 84, FORTISSIMO 110, PULSAR, MACH EQ4
```

Compressors:

```
WING COMPRESSOR, WING EXPANDER, BDX 160 COMP, BDX 560 EASY, DRAW MORE COMP, EVEN COMP/LIM,
SOUL 9000, SOUL BUS COMP, RED3 COMPRESSOR, 76 LIMITER AMP, LA LEVELER, FAIR KID, ETERNAL
BLISS, NO-STRESSOR, WAVE DESIGNER, AUTO RIDER, PIA2250 RACK, LTA100 LEVELER
```

For a wapi program to gain access to plugin parameters, independently from the plugin being installed/loaded at a given slot, the plugin parameter names are being 'anonymized' to names `1…n`, rather than the names that are listed with each single plugin. The actual parameter names for each separate plugin are listed in the plugin description tables later in this document and are preceded with their apparition number in the plugin parameter list; For example, to access the "`range`" value of plugin "`GATE`" used in channel `03`, you would set the token value to `CH_3_GATE_2`.

In the small program shown below, we replace the default **Gate** and **Dynamics** plugins for Channel 1 and set their respective parameters values to arbitrary values. For this example, we use the settings of the **AUTO RIDER DYNAMICS** gate and **Dynamics** plugins; Note that the settings are different for Gate and Compression, despite the plugin carrying the same name.

```
#include <stdio.h>
#include <string.h>
//
#include "wapi.h"
#include "wext.h"
//
int main () {

        char  wingip[24] = "";

        // we don't know the IP of our console…
        if (wOpen(wingip) != WSUCCESS) exit(1);
        printf("WING found at IP: %s\n", wingip);
        printf("Using version %i.%i\n", wVer()/256, wVer()&15);

        wSetTokenString(CH_1_GATE_MDL, "RIDE");      //Auto Rider Dynamics
        wSetTokenFloat(CH_1_GATE_1, -30.);           // thr
        wSetTokenFloat(CH_1_GATE_2, 0.);             // tgt
        wSetTokenInt(CH_1_GATE_3, 20);               // spd
        wSetTokenFloat(CH_1_GATE_4, 8.);             // ratio
        wSetTokenFloat(CH_1_GATE_5, 0.5);            // hold
        wSetTokenFloat(CH_1_GATE_6, 6.0);            // range

        wSetTokenString(CH_1_DYN_MDL, "RIDE");       //Auto Rider Dynamics
        wSetTokenFloat(CH_1_DYN_1, 50.);             // mix
        wSetTokenFloat(CH_1_DYN_2, 0.);              // gain
        wSetTokenFloat(CH_1_DYN_3, -30.);            // thr
        wSetTokenFloat(CH_1_DYN_4, 0.);              // tgt
        wSetTokenInt(CH_1_DYN_5, 20);                // spd
        wSetTokenFloat(CH_1_DYN_6, 4.);              // ratio
        wSetTokenFloat(CH_1_DYN_7, 0.5);             // hold
        wSetTokenFloat(CH_1_DYN_8, 3.0);             // range

        wClose();
        return 0;
}
```

Effects

Effects nodes are part of the main JSON structure, under the fx.n names, with n: [1…16] representing the 16 effects slots available for simultaneous use in the WIN audio processing. These 16 slots are divided in two sets of slots: 1-8 accepting premium, standard or channel effects, and slots 9-16 accepting standard and channel effects, respectively.

As in the case of plugins, the choice of effect is represented by the name (or model) of the effect, as set under the respective "mdl" token; authorized values are:

Premium

```
NONE, EXTERNAL, HALL REVERB, ROOM REVERB, CHAMBER REVERB, PLATE REVERB, CONCERT REVERB,
AMBIENCE, VSS3 REVERB, VINTAGE ROOM, VINTAGE REVERB, VINTAGE PLATE, GATED REVERB, REVERSE
REVERB, ELAY/REVERB, SHIMMER REVERB, SPRING REVERB, DIMENSION CRS, STEREO CHORUS, STEREO
FLANGER, STEREO DELAY, ULTRATAP DELAY, TAPE DELAY, OILCAN DELAYB, BD DELAY, STEREO PITCH,
DUAL PITCH
```

Standard

```
NONE, EXTERNAL, GRAPHIC EQ, PIA 560 GEQ, C5-COMBINATOR, DOUBLE VOCAL, PRECISION LIMITER,
2-BAND DEESSER, ULTRA ENHANCER, EXCITER, PSYCHO BASS, ROTARY SPEAKER, PHASER,
TREMOLO/PANNER, TAPE MACHINE, MOOD FILTER, BODYREZ, SUB OCTAVER, SUB MONSTER, PICH FIX,
RACK AMP, UK ROCK AMP, ANGEL AMP, JAZZ CLEAN AMP, DELUXE AMP, SOUL ANALOGUE, EVEN 88
FORMANT, EVEN 84, FORTISSIMO 110, PULSAR, MACH EQ4
```

Channel

```
NONE, EXTERNAL, SOUL ANALOGUE, EVEN 88 FORMANT, EVEN 84, FORTISSIMO 110, PULSAR,
MACH EQ4, EVEN CHANNEL, SOUL CHANNEL, VINTAGE CHANNEL, BUS CHANNEL, MASTERING
```

Effects can be used as dedicated inserts at defined locations within the audio path.
If an effect is part of a channel insert, assigning the effect to a different channel will remove the effect from its previous channel assignment. In order to create a more traditional effect bus, WING requires to dedicate one of the channels to the operation; Channels that want to use the effect bus can the send their audio (or a part of it) to the channel that carries the effect, creating an effect mix bus that will apply the same effect to several sources mixed into the effect channel and provide the resulting effect as a traditional effect return that can be routed to a bus.

As for the case of plugins, Effect types/engines are represented by their respective model name under the "mdl" tag, enabling the selection (loading) of a specific in one of the 16 available effect slots.

The JSON tree dedicated to effects has the following structure:

```
"fx": {
        "1": {
                "mdl": "NONE",
                "fxmix": 100
        }
        "2"…"16": {}
}
```

In fact, there are a few more, read-only[45] elements in the actual WING structure of a non-affected effect slot, resulting in the following JSON structure:

```
"fx": {
        "1": {
                "mdl": "NONE",
                "fxmix": 100,
                "$esrc": 0,     external source: [0…400]
```

[45] Read-only JSON elements start with a '$' character

```
                "$emode": M,    external mode: Mono, Stereo, Mid/Side
                "$a_chn": 0,    assign channel: [0…76]
                "$a_pos": 0     assign position: 0, 1]
        }
        "2"…"16": {}
}
```

Once an effect is assigned to a slot, the `JSON` structure for the respective slot is extended to include the parameters for the assigned effect. For example, installing reverb effect "`ROOM`" in effect slot 5 will result in the following update to the `JSON` of effect 5:

```
"fx": {
…
        "5": {
                "mdl": "ROOM",
                "fxmix": 100,
                "$esrc": 0,     [0…400]
                "$emode": M,    [M, ST, M/S]
                "$a_chn": 0,    [0, 1]
                "$a_pos": 0,    [0, 1]
                "pdel":         pre-delay
                "size":         room size
                "dcy":          decay
                "mult":         bass multiplier
                "damp":         damping
                "lc":           low cut
                "hc":           high cut
                "shp":          shape
                "sprd":         spread
                "diff":         diffusion
                "spin":         spin
                "ecl":          echo left
                "ecr":          echo right
                "efl":          feed left
                "efr":          feed right
        }
…
}
```

Each available effect is a sort of program including a set of dedicated parameters. When choosing a specific effect, the effect program is instantiated in one of the available slots and its parameters are mapped to the main Jason parameters lists for that effect slot, thus enabling for example up to 16 different copies[46] of the same effect to be active on every effect slot, with differentiated parameters for each slot.

The tables in "Appendix: Effects and Plugins' Parameters list, provide all effect names and parameters, and the parameter types associated with each known effect.

For a wapi program to gain access to effect parameters, independently from the effect being installed/loaded at a given slot, parameter names are being 'anonymized' to names `1…40`, rather than the names that are listed with each single effect. These names listed in the tables below are preceded with their apparition number in the effect parameter list; For example, to access frequency band `125Hz` of a `Graphics EQ` effect loaded at effect slot `12`, you would set the token value to `FX_12_9`.
To set/instantiate an effect in one of the 16 WING FX slots, just set the effect model's name; The effect engine will be loaded to the effect slot, discarding a previous one if there was one already. The newly installed effect parameters will become available for tweaking the effect to your settings. Model names can be found behind the tag: "`mdl`" in the tables in the "Effects and Plugins' Parameters list" appendix.

[46] For standard effects, 8 for premium effects

In the code example below is shown the instantiation of a graphic equalizer at effect slot 1, and the manipulation of its full set of parameters; In no way an interesting EQ setting, but a simple program example on how to install an effect and set parameters.

```
#include <stdio.h>
#include <string.h>
//
#include "wapi.h"
#include "wext.h"
//
int main() {
        char  wingip[24] = "";
        char *ty[] = {"std", "tru"};
        //
        // we don't know the IP of our console…
        if (wOpen(wingip)!= WSUCCESS) exit(1);
        printf("WING found at IP: %s\n", wingip);
        printf("Using version %i.%i\n", wVer()/256, wVer()&15);
        //
        int j = 0;
        float f = -15.;
        wSetTokenString(FX_1_MDL, "GEQ");
        while(1) {
                for (int I = FX_1_2; I <= FX_1_32; i++)
                        wSetTokenFloat(I, f); // bands 30Hz to 20kHz
                Sleep(1);                     // slow down!
                F += 0.25;
                if (f > 15.) {
                        f = -15;
                        j ^= 1;
                        wSetTokenString(FX_1_1, ty[j]);  // type
                }
        }
        wClose();
        return 0;
}
```

Channel strips layers

WING offer a full customization of its control surface, enabling the standard/default settings of course but any strip can be configured to become a different one.

The goal here is not to describe how to use the feature, but to warn the programmer on a specific parameter when using channel strip layers. The typical set of parameters for Section Left, layer 1 node 1 is the following:

/$ctl/layer/L/1
/$ctl/layer/L/1/ofs
/$ctl/layer/L/1/name
/$ctl/layer/L/1/1
/$ctl/layer/L/1/1/type
/$ctl/layer/L/1/1/i
/$ctl/layer/L/1/1/dst
/$ctl/layer/L/1/1/val

The type can take different values of assigned channel strip type: `CH, BUS, DCA, SEN, FX, MIDI CC`. For all but the type `MIDI CC`, the set of attributes is receiving a fixed name. In the case of `MIDI CC`, the attribute `val` is anonymized as `1`. Therefore, all wapi tokens will fit to their counterpart's name, except for `val`, as shown below:

/$ctl/layer/L/1	`$CTL_LAYER_L_1`
/$ctl/layer/L/1/ofs	`$CTL_LAYER_L_1_OFS`
/$ctl/layer/L/1/name	`$CTL_LAYER_L_1_NAME`
/$ctl/layer/L/1/1	`$CTL_LAYER_L_1_1`
/$ctl/layer/L/1/1/type	`$CTL_LAYER_L_1_1_TYPE`
/$ctl/layer/L/1/1/i	`$CTL_LAYER_L_1_1_I`
/$ctl/layer/L/1/1/dst	`$CTL_LAYER_L_1_1_DST`
/$ctl/layer/L/1/1/val	`$CTL_LAYER_L_1_1_1`

WING MIDI

WING MIDI Remote-Control channels use

WING offers a number of options and commands to remotely control the desk capabilities. Some obvious use will be for DAW control (see next Appendix: MCU [DAW BUTTONS] commands list), but more options for MIDI remote control are available:

MIDI REMOTE CONTROL mode can be used over DIN or USB WING MIDI control port .
DAW CONTROL mode can be used over DIN (limited to Control+Single MCU) or USB (Full surface) using WING MIDI DAW 1…3 ports depending on which DAW section MIDI data is coming from/going to.

Important note on USB & MIDI: Changing the clock rate or number of USB Audio channels on WING causes USB to disconnect for a few seconds (including MIDI). On certain operating systems, this may also reset already active MIDI connections. This could happen when loading snapshots with different clock rate or USB Audio configuration.

The tables below are for MIDI REMOTE CONTROL.

MIDI commands are divided by channels, most of them have been published in a Music Group Document shown below. Additional commands via MIDI CH7, CH8 and CH9 are available for Scene recall and Show control.

CC to Channel Mapping (for FADER, MUTE, PAN). FADER on MIDI CH1, MUTE on MIDI CH2, PAN on MIDI CH3:

CC12..31 → Channel 1..20
CC44..6 → Channel 21..40
CC70..77 → Aux 1..8
CC78..93 → Bus 1..16
CC94/95 → Main ½
CC102/103 → Main 3/4
CC104..111 → Matrix 1..8

DCA/Mute Groups. DCA Fader on MIDI CH4, DCA Mute and Mute Group MUTE on MIDI CH5:

CC12..27 → DCA 1..16
CC28..31 → Mute Group 1..4
CC44..47 → Mute Group 5..8

FX Parameter Control (FX1 on MIDI CH9 .. FX8 on MIDI CH16):

C12 → Insert ON/OFF
CC13 → FX Mix
CC14 → FX Model
CC15..31 → FX Parameter 1..17
CC44..58 → FX Parameter 18..32

FX Parameter Control (FX9 on MIDI CH9 .. FX16 on MIDI CH16):

CC70 → Insert ON/OFF
CC71 → FX Mix
CC72 → FX Model
CC73..95 → FX Parameter 1..23
CC102..110 → FX Parameter 24..32

Custom Controls Remote (RX only, user layers 1..16) on MIDI CH6:

CC12..15 → Layer 1 Rotaries	Note0..7→ Layer 1 Buttons (upper row, lower row)
CC16..19 → Layer 2 Rotaries	Note8..15→ Layer 2 Buttons (upper row, lower row)
CC20..23 → Layer 3 Rotaries	Note16..25→ Layer 3 Buttons (upper row, lower row)
CC24.27 → Layer 4 Rotaries	Note24..31→ Layer 4 Buttons (upper row, lower row)
CC28..31 → Layer 5 Rotaries	Note32..39→ Layer 5 Buttons (upper row, lower row)
CC44..47 → Layer 6 Rotaries	Note40..47→ Layer 6 Buttons (upper row, lower row)
CC48..51 → Layer 7 Rotaries	Note48..55→ Layer 7 Buttons (upper row, lower row)
CC52..55 → Layer 8 Rotaries	Note56..63→ Layer 8 Buttons (upper row, lower row)
CC56..59 → Layer 9 Rotaries	Note64..71→ Layer 9 Buttons (upper row, lower row)
CC60..63 → Layer 10 Rotaries	Note72..79→ Layer 10 Buttons (upper row, lower row)

CC70..73 → Layer11 Rotaries	Note80..87→ Layer 11 Buttons (upper row, lower row)
CC74..77 → Layer 12 Rotaries	Note88..95→ Layer 12 Buttons (upper row, lower row)
CC78..81→ Layer13 Rotaries	Note96..103→ Layer 13 Buttons (upper row, lower row)
CC82..85 → Layer14 Rotaries	Note104..111→ Layer 14 Buttons (upper row, lower row)
CC86..89 → Layer15 Rotaries	Note112..119→ Layer 15 Buttons (upper row, lower row)
CC90..93 → Layer16 Rotaries	Note120..127→ Layer 16 Buttons (upper row, lower row)

MIDI Scene Change (on MIDI CH7):

CC32: bank (LSB only), PC 1..128	C600..C67F

MIDI Show Control (on MIDI CH8/9):

CH8 PC 1..128 → Scene Tag #1..#128	C700..C77F
CH9 PC 1→ Scene GO	C800
CH9 PC 2→ Scene PREV	C801
CH9 PC 3→ Scene NEXT	C802
CH9 PC 4→ Scene GO + PREV	C803
CH9 PC 5→ Scene GO + NEXT	C804

WING MIDI SYSEX

WING also supports MIDI SYSEX messages, part of the MIDI protocol implementation in the console. SYSEX is a key component of MIDI implementation for advanced, digital desks as many commands are dedicated to controlling the desk as a surface control, rather than sending MIDI instrument notes. Standard 3 bytes MIDI messages are generally not long enough to support the full set of capabilities these new desks offer. This is made possible through the use and support of SYSEX functionality.

MIDI SYSEX messages are “system exclusive” data that can be passed using the MIDI HW and standard protocol to the console, using a specific formatting convention and system dedicated messages, sent over MIDI.

Your WING should be set to accept SYSEX data over `USB` or `DIN`. This setting is part of the in the `MIDI REMOTE CONTROL` tab in the `SETUP→REMOTE` screen.

SYSEX Messages format

The formatting used in SYSEX data is similar to the one used for Node Data: each parameter group is separated by a ‘,’ character, the ‘/’ character represents the root of the `JSON` parameter tree, and ‘.’ characters are used to navigate up and down within the `JSON` parameter tree, as shown below:

```
/ch.1.fdr=-1,mute=0,.2.fdr=0,mute=1
```

will set channel 1 fader to -1dB, will unmute the channel, and set channel 2 fader to 0dB and mute the channel.

SYSEX communications can report error messages (see below) that can be:

```
NODE NOT FOUND
VALUE ERROR
BUFFER OVERFLOW
```

```
NODE IS NOT PAR
INCOMPLETE DATA
STACK EMPTY
```

SYSEX Messages, Explained

WING MIDI SYSEX messages are built as follows:

```
F0 00 20 32 57 <cmd> <data[*]> F7
```

`<data[*]>` is an arbitrary number of bytes representing the data to be sent to the console, and can be an empty string of bytes; The way data is interpreted by the console is controlled using `<cmd>`, a single byte as listed below:

`00`: ident

Will return the signature string of the console, such as returned with OSC command `/?`[47] sent to port 2223, or native identification datagram `WING?`[48] Sent to port 2222. Upon executing the command, the console will return a MIDI byte `01`.

`02`: execute `<data[*]>`

Will execute the requested command contained in or represented by `<data[*]>`. `<data[*]>` should contain a WING command respecting the format rules described earlier in this chapter. Upon executing the command, the console may return status `07 (Error)` followed with an error message if an error occurred.

`03`: dump `<data[*]>`

Will return the current values found for `<data[*]>`. `<data[*]>` should contain a WING node or command respecting the format rules described earlier in this chapter. Upon executing the command, the console will return MIDI byte `04` (OK) or `07 (Error)` followed with an error message.

`05`: describe `<data[*]>`

Will return the description of the node found at or represented by `<data[*]>`. `<data[*]>` should contain a WING node respecting the format rules described earlier in this chapter. Upon executing the command, the console will return a MIDI byte `06` (OK) or `07` followed with an error message.

Examples

We give below a few examples of MIDI SYSEX commands sent to WING, with their interpretation, their resulting effect on WING, and the returned data, if any.

cmd = 00 example:

Sending `F0 00 20 32 57 00 f7`,
or `cmd 00` will generate a SYSEX being returned by WING:

```
F0 00 20 32 57 01 57 49 4E 47 2C 31 39 32 2E 31 36 38
2E 31 2E 37 31 2C 50 47 4D 2C 6E 67 63 2D 66 75 6C 6C
2C 4E 4F 5F 53 45 52 49 41 4C 2C 31 2E 30 38 2E 31 2D
```

[47] Refer to chapter on OSC protocol
[48] Refer to chapter "Remote communications with WING"

```
30 2D 67 33 33 65 36 39 66 38 38 3A 72 65 6C 65 61 73
65 F7
```

Containing the WING signature *(your system will contain something slightly different)*:
`WING,192.168.1.71,PGM,ngc-full,NO_SERIAL,1.08.1-0-g33e69f88:release`

cmd = 02 examples:

Sending `F0 00 20 32 57 02 2F 63 68 2E 31 2E 66 64 72 3D 2D 31 2C 6D 75 74 65 3D 30 2C 2E 32 2E 66 64 72 3D 30 2C 6D 75 74 65 3D 31 F7`,
or `cmd 02` followed by `/ch.1.fdr=-1,mute=0,.2.fdr=0,mute=1` will set channel 1 fader to -1, channel 2 fader to 0 and will unmute channel 1 and mute channel 2.

Sending `F0 00 20 32 57 02 2F 63 68 2E 31 2E 66 64 3D 2D 31 2C 6D 75 74 65 3D 30 2C 2E 32 2E 66 64 72 3D 30 2C 6D 75 74 65 3D 31 F7`,
or `cmd 02` followed by `/ch.1.fd=-1,mute=0,.2.fdr=0,mute=1` will return an error; note we omitted the "r" of "fdr" above for channel 1. WING will reply with the following SYSEX message: `F0 00 20 32 57 07 4E 4F 44 45 20 4E 4F 54 20 46 4F 55 4E 44 F7`, or error status `07`, followed by `NODE NOT FOUND.`

cmd = 03 examples:

Sending `F0 00 20 32 57 03 2f 61 75 78 F7`,
or `cmd 03` followed by `/aux`, will be replied by WING with a near 28000 bytes SYSEX message corresponding to the following in ASCII (partial listing containing `aux1` and `aux8, aux2` to `aux7` included but not listed):

```
1.in.set.srcauto=0,altsrc=0,inv=0,trim=0.0,bal=0.0,.conn.grp=USB,in=1,altgrp=OFF,altin=1,..
clink=0,col=8,name=USB,icon=605,led=1,mute=0,fdr=oo,pan=0,wid=100,solosafe=0,mon=A,eq.on=0,
mix=100,lg=0.0,lf=80.2,lq=2.00,leq=SHV,1g=0.0,1f=399.1,1q=2.00,2g=0.0,2f=2k50,2q=2.00,hg=0.
0,hf=11k99,hq=2.00,heq=SHV,.preins.on=0,ins=NONE,.main.1.on=1,lvl=0.0,.2.on=0,lvl=0.0,.3.on
=0,lvl=0.0,.4.on=0,lvl=0.0,..send.1.on=0,lvl=oo,pon=0,ind=0,mode=PRE,plink=0,pan=0,wid=100,
.2.on=0,lvl=oo,pon=0,ind=0,mode=PRE,plink=0,pan=0,wid=100,.3.on=0,lvl=oo,pon=0,ind=0,mode=P
RE,plink=0,pan0,wid=100,.4.on=0,lvl=oo,pon=0,ind=0,mode=PRE,plink=0,pan=0,wid=100,.5.on=0,l
vl=oo,pon=0,ind=0,mode=PRE,plink=0,pan=0,wid=100,.6.on=0,lvl=oo,pon=0,ind=0,mode=PRE,plink=
0,pan=0,wid=100,.7.on=0,lvl=oo,pon=0,ind=0,mode=PRE,plink=0,pan=0,wid=00,.8.on=0,lvl=oo,pon
=0,ind=0,mode=PRE,plink=0,pan=0,wid=100,.9.on=0,lvl=oo,pon=0,ind=0,mode=PRE,plink=0,pan=0,w
id=100,.10.on=0,lvl=oo,pon=0,ind=0,mode=PRE,plink=0,pan=0,wid=100,.11.on=0,lvl=oo,pon=0,ind
=0,mode=PRE,plink=0,pan=0,wid=100,.12.on=0,lvl=oo,pon=0,ind=0,mode=PRE,plink=0,pan=0,wd=100
,.13.on=0,lvl=oo,pon=0,ind=0,mode=PRE,plink=0,pan=0,wid=100,.14.on=0,lvl=oo,pon=0,ind=0,mod
e=PRE,plink=0,pan=0,wid=100,.15.on=0,lvl=oo,pon=0,ind=0,mode=PRE,plink=0,pan=0,wid=100,.16.
on=0,lvl=oo,pon=0,ind=0,mode=PRE,plink=0,pan=0,wid100,..tags=,.2.in.set.srcauto=0,[…Aux cha
nnels 2 to 7 listed…],..tags=.8.in.set.srcauto=0,altsrc=0,inv=0,trim=0.0,bal=0.0,.conn.grp=
OFF,in=1,altgrp=OFF,altin=1,..clink=0,col=1,name=,icon=0,led=1,mute=0,fdr=oo,pan=0,wid=100,
solosafe=0,mon=A,eq.on=0,mix=100,lg=0.0,lf=80.2,lq=2.00,leq=SHV,1g=0.0,1f=399.1,1q=2.00,2g=
0.0,2f=2k50,2q=2.00,hg=0.0,hf=11k99,hq=2.00,heq=SHV,.preins.on=0,ins=NONE,.main.1.on=1,lvl=
0.0,.2.on=0,lvl=0.0,.3.on=0,lvl=0.0,.4.on=0,lvl=0.0,..send.1.on=0,lvl=oo,pon=0,ind=0,mode=P
RE,plink=0,pan=0,wid=100,.2.on=0,lvl=oo,pon=0,ind=0,mode=PRE,plink=0,pan=0,wid=100,.3.on=0,
lvl=oo,pon=0,ind=0,mode=PRE,plink=0,pan=0,wid=100,.4.on=0,lvl=oo,pon=0,ind=0,mode=PRE,plink
=0,pan=0,wid=100,.5.on=0,lvl=oo,pon=0,ind=0,mode=PRE,plink=0,pan=0,wid=100,.6.on=0,lvl=oo,p
on=0,ind=0,mode=PRE,plink=0,pan=0,wid=100,.7.on=0,lvl=oo,pon=0,ind0,mode=PRE,plink=0,pan=0,
wid=100,.8.on=0,lvl=oo,pon=0,ind=0,mode=PRE,plink=0,pan=0,wid=100,.9.on=0,lvl=oo,pon=0,ind=
0,mode=PRE,plink=0,pan=0,wid=100,.10.on=0,lvl=oo,pon=0,ind=0,mode=PRE,plink=0,pan=0,wid=100
,.11.on=0,lvl=oo,pon=0,ind=0,mode=PRE,plink=0,pan=0,wid=100,.12.on=0,lvl=oo,pon=0,ind=0,mod
e=PRE,plink=0,pan=0,wid=100,.13.on=0,lvl=oo,pon=0,ind=0,mode=PRE,plink=0,pan=0,wid=100,.14.
on=0,lvl=oo,pon=0,ind=0,mode=PRE,plink=0,pan=0,wid=100,.15.on=0,lvl=oo,pon=0,ind=0,mode=PRE
,plink=0,pan=0,wid=100,.16.on=0,lvl=oo,pon=0,ind=0,mode=PRE,plink=0,pan=0,wid=100,..tags=,.
```

Sending `F0 00 20 32 57 03 2F 63 68 2E 31 2E 66 64 72 F7`,

or `cmd 03` followed by `/ch.1.fdr`, will be replied by WING with a SYSEX message containing the current value of channel 1 fader: `F0 00 20 32 57 04 2D 31 2E 30 F7`, or status `04` (no error) and value `-1.0`.

cmd = 05 examples:

Sending again SYSEX `F0 00 20 32 57 05 2f 61 75 78 00 00 00 00 F7`,
Or `cmd 05` followed by `/aux~~~~` will be replied by WING with the following SYSEX message (note the status `06` (no error):

```
F0 00 20 32 57 06 20 20 31 20 20 20 20 20 20 20 20
20 20 20 20 20 28 6E 6F 64 65 29 0A 20 20 32 20 20 20
20 20 20 20 20 20 20 20 20 20 20 28 6E 6F 64 65 29 0A
20 20 33 20 20 20 20 20 20 20 20 20 20 20 20 20 28
6E 6F 64 65 29 0A 20 20 34 20 20 20 20 20 20 20 20 20
20 20 20 20 28 6E 6F 64 65 29 0A 20 20 35 20 20 20
20 20 20 20 20 20 20 20 20 20 20 28 6E 6F 64 65 29 0A
20 20 36 20 20 20 20 20 20 20 20 20 20 20 20 20 28
6E 6F 64 65 29 0A 20 20 37 20 20 20 20 20 20 20 20 20
20 20 20 20 28 6E 6F 64 65 29 0A 20 20 38 20 20 20
20 20 20 20 20 20 20 20 20 20 20 28 6E 6F 64 65 29 0A
F7
```

Or in ASCII:

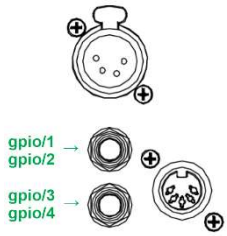
```
1               (node)
2               (node)
3               (node)
4               (node)
5               (node)
6               (node)
7               (node)
8               (node)
```

Sending again `F0 00 20 32 57 05 2F 63 68 2E 31 2E 66 64 72 F7`,
or `cmd 05` followed by `/ch.1.fdr`, will be replied by WING with a SYSEX message containing the description of channel 1 fader: `F0 00 20 32 57 06 66 61 64 65 72 20 5B 2D 6F 6F 20 2E 2E 20 31 30 2E 30 20 64 42 5D 2C 20 31 30 32 34 20 73 74 65 70 73 0A F7`, or status `06` (no error), followed by description `fader [-oo .. 10.0 dB], 1024 steps`.

Appendices

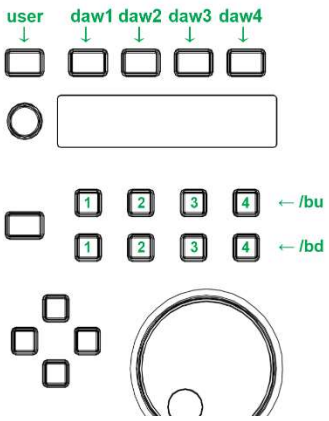
Appendix: Buttons (user/gpio, user/user, user/daw, user/)

WING includes a rather large set of buttons separated in different logical blocks: user/gpio, user/user, and user/daw and user. They are all managed under the `$ctl` subtree of commands. As in the case of effects where the effect model sets the type and number of OSC patterns available for supporting the functionality currently in effect, the associated `JSON` structure varies and adapts to the necessary sets of parameters.



user/gpio/1..4

This subsection covers the 4 possible GPIOs supported by WING; the actual set of usable OSC patterns available at a given time depends on the `mode` parameter value of the `/$ctl/user/gpio/1..4/bu/` OSC pattern represented below as `<OSC b_pattern>`.

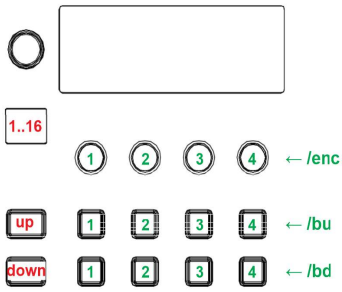


user/user/1..4

This subsection covers the 8 user buttons supported by WING; the actual set of usable OSC patterns available at a given time depends on the `mode` parameter value of the `/$ctl/user/user/1..4/bu/` and the `/$ctl/user/user/1..4/bd/` OSC patterns for the 4 buttons of the upper and lower row of the button section, and represented below as `<OSC b_pattern>`.

user/daw1..4/1..4

This subsection covers the 4 possible sets of 8 DAW buttons supported by WING; the actual set of usable OSC patterns available at a given time depends on the `mode` parameter value of the `/$ctl/user/daw1..daw4/1..4/bu/` and the `/$ctl/user/daw1..daw4/1..4/bd/` OSC patterns for the 4 buttons of the upper and lower row of the button section, and represented below as `<OSC b_pattern>`.



user/1..16/1..4

This subsection covers the 16 possible sets of 8 user buttons and 4 user encoders supported by WING; the actual set of usable OSC patterns available at a given time depends on the `mode` parameter value of the `/$ctl/user/1..16/1..4/bu/`, `/$ctl/user/1..16/1..4/bd/`, and `/$ctl/user/1..16/1..4/enc/` OSC patterns for the 4 buttons of the upper and lower row of the button section, and the 4 encoders represented below as `<OSC b_pattern>` and `<OSC e_pattern>`, respectively.

The tables below list the different options for OSC patterns `<OSC b_pattern>`:

mode	Command	Type	Range / Text	Description
OFF	none			OFF
MUTE	<OSC b_pattern>/ch	I	1..76	Channel number
INS1	<OSC b_pattern>/ch	I	1..76	Channel number
INS2	<OSC b_pattern>/ch	I	1..76	Channel number
MGRP	<OSC b_pattern>/mgrp	S	MGRP.1, MGRP.2..MGRP.8	Mute group number
DCAMUTE	<OSC b_pattern>/dca	S	DCA.1, DCA.2..DCA.16	DCA fader number mute
SOF	<OSC b_pattern>/ch	I	1..76	Channel number
SPILL	<OSC b_pattern>/area	S	L, C, R	Console area (left, center, right)
	<OSC b_pattern>/tgt	S	DCA1,.., DCA16, FX1,.., FX16, BUS1,.., BU16, AUTOX, AUTOY	Targeted group
FXPAR	<OSC b_pattern>/fx	S	FX1..FX16	FX processor number
	<OSC b_pattern>/par	I	1..41	FX processor parameter number[49]
DAWBTN	<OSC b_pattern>/btn	S	T1..T20, N1..N9, A1..A16, F1..F8, V1..V15, AU1..AU12, SY1..SY10, OT1..OT12, E1..E10, SP1.., SP6	MCU button[50]
DAWENC	<OSC b_pattern>/enc	S	M1P..M8P, E1P..E16P, M1..M8, E1..E16, JOG	DAW Rotary[51]
CHPAGE	<OSC b_pattern>/ch	I	1..76	Channel number
	<OSC b_pattern>/pg	S	HOME, INPUT, FILT, GATE, DYN, EQ, INS1, INS2, MAIN, SEND, SND.CFG, SND.EQ	Page name
PAGE	<OSC b_pattern>/pg	S	FX, MTRS, CHINS, SRC, OUTS, SETUP, LIB, CUSTCTL, MON, 2TRK, WLIVE, MIXV, FDRV, SENDV, MGRP, LAYER	Page name
FDRPAGE	<OSC b_pattern>/area	S	L, C, R	Area
	<OSC b_pattern>/bank	I	1..19	Bank
VIEWPAGE	<OSC b_pattern>/area	S	L, C, R	Area
	<OSC b_pattern>/bank	I	1..19	Bank
OTHER	<OSC b_pattern>/other	S	TBA, TBB, ALTSRC, DAWSW, MONA, MONB, MONAB, MONDIM, MONMONO,	Other functions

[49] 1..40 are for FX parameters, 41 is for FX Mix
[50] See MCU [DAW BUTTONS] commands list in Appendixes
[51] See MCU [DAW V-POTS] commands list in Appendixes

			MONSWAP, MONMUTE, FDROFF, FDR0DB, AUTOX, AUTOY	
GPIO	<OSC b_pattern>/GPIO	S	A, B, C, D, A-P, B-P, C-P, D-P,	GPIO Toggle or Push
FSTART	<OSC b_pattern>/ch	I	1..76	Channel number
MIDICCT	<OSC b_pattern>/ch	I	1..16	MIDI channel (toggle)
	<OSC b_pattern>/cc	I	0..127	MIDI control change number
	<OSC b_pattern>/val	I	0..127	MIDI control value
MIDICCP	<OSC b_pattern>/ch	I	1..16	MIDI channel (push)
	<OSC b_pattern>/cc	I	0..127	MIDI control change number
	<OSC b_bpattern>/val	I	0..127	MIDI control value
MIDINT	<OSC b_pattern>/ch	I	1..16	MIDI channel (toggle)
	<OSC b_bpattern>/note	I	0..127	MIDI note
	<OSC bb_pattern>/val	I	0..127	MIDI note value
MIDINP	<OSC b_pattern>/ch	I	1..16	MIDI channel (push)
	<OSC b_pattern>/note	I	0..127	MIDI note
	<OSC b_pattern>/val	I	0..127	MIDI note value
MIDIPGM	<OSC b_pattern>/ch	I	1..16	MIDI channel
	<OSC b_pattern>/note	I	1..128	MIDI program value
USBPR	<OSC b_pattern>/usbpr	S	PSTOP, PLAY, PPAUSE, PNEXT, PPREV, RSTOP, RECORD, RPAUSE, RNEW	USB Play Rec
SDRECA	<OSC b_pattern>/sdrec	S	STOP, PLAY, REC, PAUSE, PLAYSTOP, PLAYPAUSE, ADD, PREV, NEXT, PLAYMARKER, GOMARKER, SELSESSION, PREV_S, NEXT_S	SD A recorder
SESSIONA	<OSC b_pattern>/session	S	S1..S20	SD A Session
MARKERA	<OSC b_pattern>/marker	S	M1..M20	SD A Marker
SDRECB	<OSC b_pattern>/sdrec	S	STOP, PLAY, REC, PAUSE, PLAYSTOP, PLAYPAUSE, ADD, PREV, NEXT, PLAYMARKER, GOMARKER, SELSESSION, PREV_S, NEXT_S	SD B recorder
SESSIONB	<OSC b_pattern>/session	S	S1..S20	SD B Session
MARKERB	<OSC b_pattern>/marker	S	M1..M20	SD B Marker

The table below lists the different options for the OSC encoder pattern `<OSC e_pattern>`:

mode	Command	Type	Range / Text	Description
OFF	none			OFF
FDR	<OSC e_pattern>/ch	I	1..76	Channel number
PAN	<OSC e_pattern>/ch	I	1..76	Channel number
DCA	<OSC e_pattern>/dca	S	DCA.1, DCA.2..DCA.16	DCA fader number
SSND	<OSC e_pattern>/send	S	BUS1, …, BUS16	Send to Bus number
FSND	<OSC e_pattern>/ch <OSC e_pattern>/send	I S	1..48 BUS1, …, BUS16	Channel number Send to Bus number
FX	<OSC e_pattern>/fx	S	FX1..FX16	FX processor number
	<OSC e_pattern>/par	I	1..41	FX processor parameter number[52]
DAWMCU	<OSC e_pattern>/mcuenc	S	M1..M8, E1..E16, JOG	DAW Rotary[53]
MON	<OSC e_pattern>/mon	S	A, B	Monitor selection[54] A: PHONES level B: SPEAKERS / MONITOR level
MIDICC	<OSC e_pattern>/ch	I	1..16	MIDI channel
	<OSC e_pattern>/cc	I	0..127	MIDI control change number
	<OSC e_pattern>/val	I	0..127	MIDI control change value
SD A	<OSC e_pattern>/sdarec	S	POS, MARKER, SESSION	SD-A Recorder
SD B	<OSC e_pattern>/sdbrec	S	POS, MARKER, SESSION	SD-B Recorder

[52] 1..40 are for FX parameters, 41 is for FX Mix

[53] See MCU [DAW REMOTE MCU] commands list in Appendixes

[54] Note these are only 'readable' when touching the encoder, the actual setting is only possible with the potentiometer in the Monitoring/Talkback section of the console

Appendix: Effects and Plugins' Parameters list

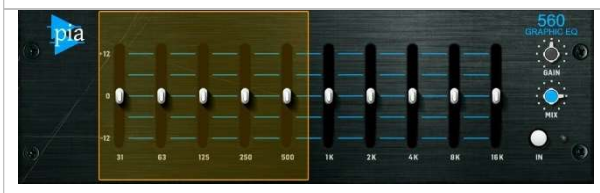
In the (long) tables below, we list all known/exposed effects and plugins available with the WING digital console, along with their name, type, and min/max/step/list values; We therefore present Standard Effects, Premium effects, Filter Plugins, Gate Plugins, EQ Plugins, and Compressor Plugins.
All active effects and plugins modify the `JSON` tree and their respective OSC patterns. The tables below show all parameters associated to effects and plugins, including their name, type, and value range following the OSC pattern `/fx/1..16/`

On any channel, an insert will create a processing delay of 32 samples (i.e. around 0.66ms). This is because audio is routed to a different DSP for FX processing. It is important to take this into account when mixing, as phasing effects may result from the imposed delay.

Effects

Standard effects

	None *0 "mdl": NONE*
	*External* *0 "mdl": EXT* *1 "egrp": str [OFF, LCL, AUX, A, B, C, SC, USB, CRD, MOD, PLAY, AES] ext grp* *2 "ein": int [1…64] ext in* *3 "emode": str [M, ST, M/S] ext mode* *4 "lat": int [0…200] latency* *5 "trim": linf [-18, 18, 361] dB, trim*
	*Graphic EQ* *0 "mdl": GEQ* *1 "type": str [STD, TRU] geq type* *2 "20": linf [-15, 15, 121] dB* *3 "25": linf [-15, 15, 121] dB* *4 "31": linf [-15, 15, 121] dB* *5 "40": linf [-15, 15, 121] dB* *6 "50": linf [-15, 15, 121] dB* *7 "63": linf [-15, 15, 121] dB* *8 "80": linf [-15, 15, 121] dB* *9 "100": linf [-15, 15, 121] dB* *10 "125": linf [-15, 15, 121] dB* *11 "160": linf [-15, 15, 121] dB* *12 "200": linf [-15, 15, 121] dB* *13 "250": linf [-15, 15, 121] dB* *14 "315": linf [-15, 15, 121] dB* *15 "400": linf [-15, 15, 121] dB* *16 "500": linf [-15, 15, 121] dB* *17 "630": linf [-15, 15, 121] dB* *18 "800": linf [-15, 15, 121] dB* *19 "1k": linf [-15, 15, 121] dB* *20 "1k25": linf [-15, 15, 121] dB* *21 "1k6": linf [-15, 15, 121] dB* *22 "2k": linf [-15, 15, 121] dB* *23 "2k5": linf [-15, 15, 121] dB* *24 "3k15": linf [-15, 15, 121] dB* *25 "4k": linf [-15, 15, 121] dB* *26 "5k": linf [-15, 15, 121] dB* *27 "6k3": linf [-15, 15, 121] dB* *28 "8k": linf [-15, 15, 121] dB* *29 "10k": linf [-15, 15, 121] dB* *30 "12k5": linf [-15, 15, 121] dB* *31 "16k": linf [-15, 15, 121] dB*

	32 **“20k”**: linf [-15, 15, 121] dB
	PIA 560 GEQ 0 **“mdl”**: PIA 1 **“mix”**: linf [0, 125, 126] %, mix 2 **“gain”**: linf [-12, 12, 241] dB 3 **“31”**: linf [-12, 12, 241] dB 4 **“63”**: linf [-12, 12, 241] dB 5 **“125”**: linf [-12, 12, 241] dB 6 **“250”**: linf [-12, 12, 241] dB 7 **“500”**: linf [-12, 12, 241] dB 8 **“1k”**: linf [-12, 12, 241] dB 9 **“2k”**: linf [-12, 12, 241] dB 10 **“4k”**: linf [-12, 12, 241] dB 11 **“8k”**: linf [-12, 12, 241] dB 12 **“16k”**: linf [-12, 12, 241] dB
	Triple Dynamic EQ 0 **“mdl”**: DEQ3 1 **“1-thr”**: linf [-60, 0, 121] dB, threshold 1 2 **“1-ratio”**: str [1.20, 1.30, 1.50, 2.00, 3.00, 5.00, 10.00] ms, ratio 1 3 **“1-att”**: linf [0.00, 200.00, 201] ms, att 1 4 **“1-rel”**: logf [20.00, 4000.00, 130] ms, rel 1 5 **“1-filt”**: str [OFF, BP, LP6, LP12, HP6, HP12] 6 **“1-g”**: linf [-15.00, 15.00, 301] dB, gain 1 7 **“1-f”**: logf [20, 20000, 961] Hz, freq 1 8 **“1-q”**: logf [0.44, 10.00, 181] qual 1 9 **“1-mode”**: str [low, high] mode 1 10 **“2-thr”**: linf [-60, 0, 121] dB, threshold 2 11 **“2-ratio”**: str [1.20, 1.30, 1.50, 2.00, 3.00, 5.00, 10.00] ms, ratio 2 12 **“2-att”**: linf [0.00, 200.00, 201] ms, att 2 13 **“2-rel”**: logf [20.00, 4000.00, 130] ms, rel 2 14 **“2-filt”**: str [OFF, BP, LP6, LP12, HP6, HP12] 15 **“2-g”**: linf [-15.00, 15.00, 301] dB, gain 2 16 **“2-f”**: logf [20, 20000, 961] Hz, freq 2 17 **“2-q”**: logf [0.44, 10.00, 181] qual 2 18 **“2-mode”**: str [low, high] mode 2 19 **“3-thr”**: linf [-60, 0, 121] dB, threshold 3 20 **“3-ratio”**: str [1.20, 1.30, 1.50, 2.00, 3.00, 5.00, 10.00] ms, ratio 3 21 **“3-att”**: linf [0.00, 200.00, 201] ms, att 3 22 **“3-rel”**: logf [20.00, 4000.00, 130] ms, rel 3 23 **“3-filt”**: str [OFF, BP, LP6, LP12, HP6, HP12] 24 **“3-g”**: linf [-15.00, 15.00, 301] dB, gain 3 25 **“3-f”**: logf [20, 20000, 961] Hz, freq 3 26 **“3-q”**: logf [0.44, 10.00, 181] qual 3 27 **“3-mode”**: str [low, high] mode 3
	Combinator 0 **“mdl”**: C5-CMB 1 **“thr”**: linf [-40, 0, 401] dB, threshold 2 **“gain”**: linf [-10, 10, 201] dB, gain 3 **“ratio”**: str [1.1, 1.2, 1.3, 1.5, 1.7, 2.0, 2.5, 3.0, 3.5, 4.0, 5.0, 7.0, 10.0, 100.0] ms, ratio 4 **“slope”**: str [24, 48] dB/Oct, slope 5 **“bandse l”**: int [1..5] selected band 6 **“att”**: linf [0, 20, 21] attack 7 **“rel”**: logf [20, 3000, 201] ms, release 8 **“arel”**: int [0, 1] auto release 9 **“sbc”**: linf [1, 10, 10] sbc speed 10 **“sbcon”**: int [0, 1] sbc on 11 **“thr_1”**: linf [-10, 10, 201] dB, 1-THR 12 **“thr_2”**: linf [-10, 10, 201] dB, 2-THR 13 **“thr_3”**: linf [-10, 10, 201] dB, 3-THR 14 **“thr_4”**: linf [-10, 10, 201] dB, 4-THR 15 **“thr_5”**: linf [-10, 10, 201] dB, 5-THR 16 **“gain_1”**: linf [-10, 10, 201] dB, 1-GAIN 17 **“gain_2”**: linf [-10, 10, 201] dB, 2-GAIN

	18 **"gain_3"**: linf [-10, 10, 201] dB, 3-GAIN 19 **"gain_4"**: linf [-10, 10, 201] dB, 4-GAIN 20 **"gain_5"**: linf [-10, 10, 201] dB, 5-GAIN 21 **"byp_1"**: int [0, 1], 1-BYP 22 **"byp_2"**: int [0, 1], 2-BYP 23 **"byp_3"**: int [0, 1], 3-BYP 24 **"byp_4"**: int [0, 1], 4-BYP 25 **"byp_5"**: int [0, 1], 5-BYP 26 **"width_1"**: linf [-50, 50, 101], 1-XOVER 27 **"width_2"**: linf [-50, 50, 101], 2-XOVER 28 **"width_3"**: linf [-50, 50, 101], 3-XOVER 29 **"width_4"**: linf [-50, 50, 101], 4-XOVER 30 **"width_5"**: linf [-50, 50, 101], 5-XOVER 31 **"mix"**: linf [0, 100, 101], mix 32 **"$bdsolo"**: int [0, 1] band solo
	Precision Limiter 0 **"mdl"**: LIMITER 1 **"gin"**: linf [0, 18, 73] dB, in gain 2 **"gout"**: linf [-18, 0, 73] dB out gain 3 **"sqz"**: int [0…100] sqeeze 4 **"knee"**: int [0…10] knee 5 **"again"**: int [0, 1] auto gain 6 **"att"**: linf [.05, 1, 95] ms, attack 7 **"rel"**: logf [20, 2000, 101] ms, release
	Speaker Manager 0 **"mdl"**: SPKMAN 1 **"hpf"**: logf [20.00, 20000.00, 961] Hz, high 2 **"hptype"**: str [FLAT, BW6, BW12, BS12, LR12, BW18, BW24, BS24, LR24, BW48, LR48] type 3 **"lpf"**: logf [20.00, 20000.00, 961] Hz, low 4 **"lptype"**: str [FLAT, BW6, BW12, BS12, LR12, BW18, BW24, BS24, LR24, BW48, LR48] type 5 **"tiltf"**: logf [100.00, 10000.00, 121] Hz, tilt 6 **"tiltg"**: linf [-6.00, 6.00, 121] dB, tilt gain 7 **"phase"**: linf [0.00, 180.00, 37] phase 8 **"invert "**: int [0, 1] invert 9 **"dist"**: linf [0.00, 5.00, 501] mtrs, distance 10 **"pos"**: linf [-5.00, 5.00, 1001] mtrs, pos. 11 **"dyneq"**: int [0, 1] deq 12 **"dynthr"**: linf [-60.00, 0.00, 121] dB, threshold 13 **"deqratio"**: str [1.20, 1.30, 1.50, 2.00, 3.00, 5.00, 10.00] ratio 14 **"deqatt"**: linf [0.00, 200.00, 201] ms, attack 15 **"deqrel"**: logf [20.00, 4000.00, 130] ms, release 16 **"deqfilt"**: str [OFF, BP, LP6, LP12, HP6, HP12] 17 **"deqg"**: linf [-15.00, 15.00, 301] dB, gain 18 **"deqf"**: logf [20.00, 20000.00, 961] Hz, freq 19 **"deqq"**: logf [0.44, 10.00, 181] qual 20 **"deqmode"**: str [low, high] mode 21 **"lim"**: int [0, 1] limiter 22 **"limthr"**: linf [-24.00, 0.00, 241] dB, threshold 23 **"limrms"**: int [0, 1] rms 24 **"limrel"**: logf [50.00, 2000.00, 121] ms, release
	2-Band DeEsser 0 **"mdl"**: DE-S2 1 **"lo"**: linf [0, 50, 51] low 2 **"hi"**: linf [0, 50, 51] high 3 **"los"**: linf [0, 50, 51] low (s) 4 **"his"**: linf [0, 50, 51] high (s) 5 **"gdr"**: str [FEMALE, MALE] gender 6 **"mode"**: str [STEREO, MID/SIDE] mode

Ultra Enhancer

```
 0 "mdl":   ENHANCE
 1 "stlv":  linf [-100, 100, 201] %, st lvl
 2 "lmf":   linf [-100, 100, 201] %, lmf spread
 3 "lmvl":  linf [-100, 100, 201] %, mono lvl
 4 "st":    linf [-100, 100, 201] %, st pan
 5 "m":     linf [-100, 100, 201] %, mono pan
 6 "bass":  linf [0, 100, 101] %, bass gain
 7 "mid":   linf [0, 100, 101] %, mid gain
 8 "high":  linf [0, 100, 101] %, high gain
 9 "g":     linf [-112, 12, 241] dB, gain
10 "solo":  int [0, 1] solo
11 "bassf": linf [1, 50, 50] bass freq
12 "midq":  linf [1, 50, 50] mid Q
13 "highf": linf [1, 50, 50] high freq
```

Exciter

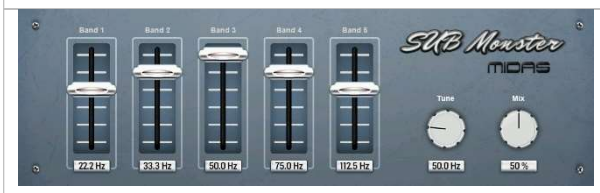
```
 0 "mdl":   EXCITER
 1 "tune":  logf [1000, 10000, 51] Hz, tune
 2 "peak":  linf [0, 100, 101] %, peak
 3 "zfill": linf [0, 100, 101] %, zfill
 4 "timbre":linf [-50, 50, 101] timbre
 5 "harm":  linf [0, 100, 101] %, harm
 6 "mix":   linf [0, 100, 101] %, mix
 7 "dry":   int [0, 1] dry
```

Psycho Bass

```
 0 "mdl":  P-BASS
 1 "int":  linf [-24, 6, 61] dB, intensity
 2 "bass": linf [-60, 0, 121] dB, bass gain
 3 "xf":   logf [32, 200. 51] Hz, X/O freq
 4 "solo": int [0, 1] solo
```

Sub Octaver

```
 0 "mdl":   SUB
 1 "rng":   str [LOW, MID, HIGH] range
 2 "oct1":  linf [0, 100, 101] %, octave 1
 3 "oct2":  linf [0, 100, 101] %, octave 2
```

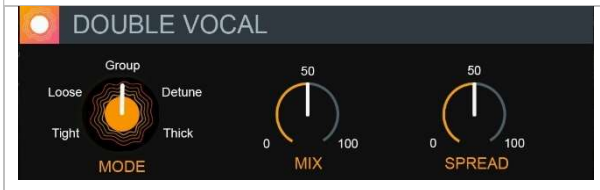


Sub Monster

```
 0 "mdl":   SUB-M
 1 "mix":   [0, 100, 101] %mix
 2 "freq":  linf [45, 67.5, 226] Hz, freq
 3 "bd1":   linf [0, 100, 101] %, band 1
 4 "bd2":   linf [0, 100, 101] %, band 2
 5 "bd3":   linf [0, 100, 101] %, band 3
 6 "bd4":   linf [0, 100, 101] %, band 4
 7 "bd5":   linf [0, 100, 101] %, band 5
```

Velvet Imager

```
 0 "mdl":   V_IMG
 1 "wid":   linf> [-1.00, 1.00, 201] width
 2 "st":    linf [0.00, 100.00, 101] %, stereo
 3 "gain":  linf [-6.00, 6.00, 49] dB, gain
 4 "mode":  str [K, VELVET] mode
 5 "deep":  int [0,1] deep
```



Double Vocal

```
 0 "mdl":  DOUBLE
 1 "mode": str [TIGHT, LOOSE, GROUP,
                DETUNE, THICK] mode
 2 "mix":  linf [0, 100, 101] %, mix
 3 "sprd": linf [0, 100, 101] %, spread
```

Pitch Fix

```
 0 "mdl":  PCORR
 1 "spd":  linf [1, 100, 100] speed
 2 "amnt": linf [0, 50, 51] amount
 3 "a4":   linf [410, 470, 601] A4 pitch
 4 "_c":   int [0, 1]
 5 "_db":  int [0, 1]
 6 "_d":   int [0, 1]
 7 "_eb":  int [0, 1]
 8 "_e":   int [0, 1]
 9 "_f":   int [0, 1]
10 "_gb":  int [0, 1]
11 "_g":   int [0, 1]
12 "_ab":  int [0, 1]
13 "_a":   int [0, 1]
14 "_bb":  int [0, 1]
15 "_b":   int [0, 1]
```

Rotary Speaker

```
 0 "mdl":  ROTARY
 1 "sw":   str [STOP, SLOW, FAST]
 2 "lo":   logf [.1, 3.999, 51] Hz, lo speed
 3 "hi":   logf [4, 10, 51] Hz, hi speed
 4 "bal":  linf [-100, 100, 201] balance
 5 "mix":  linf [0, 100, 101] %, mix
 6 "dist": linf [0, 100, 101] distance
 7 "dac":  linf [0, 100, 101] %, drum accel
 8 "hac":  linf [0, 100, 101] %, horn accel
```

Phaser

```
 0 "mdl":    PHASER
 1 "spd":    logf [.05, 5, 201] Hz, speed
 2 "phase":  int [0…180] phase
 3 "wave":   int [-50…50] wave
 4 "range":  int [2…98] %, range
 5 "depth":  int [0…100] %, depth
 6 "emod":   int [-100, 100] % env mod
 7 "att":    logf [10, 1000, 201] ms, attack
 8 "hld":    logf [10, 2000, 201] ms, hold
 9 "rel":    logf [10, 1000, 201] ms, release
10 "mix":    int [0…100] %, mix
11 "stg":    int [2…12] stages
12 "reso":   int [0…80] %, reso
```

Tremolo Panner

```
 0 "mdl":    PANNER
 1 "att":    logf [10, 1000, 201] ms, attack
 2 "hld":    logf [10, 2000, 201] ms, hold
 3 "rel":    logf [10, 1000, 201] ms, release
 4 "espd":   int [0…100] %, env>depth
 5 "edep":   int [0…100] %, env>depth
 6 "spd":    logf [.05, 5, 201] Hz, speed
 7 "phase":  int [0…180] phase
 8 "wave":   int [-50…50] wave
 9 "depth":  int [0…100] %, depth
```

Tape Machine

```
 0 "mdl":    TAPE
 1 "drv":    linf [-12, 12, 97] dB, drive
 2 "spd":    logf [7.5, 30, 65]
 3 "low":    int [0, 1] low bump
 4 "hi":     int [0, 1] high shelv
 5 "out":    linf [-12, 12, 97] dB, out gains s
```

Mood Filter

```
 0 "mdl":    MOOD
 1 "fbase":  logf [20, 15000, 101] Hz, base
 2 "filt":   str [LP, HP, BP, NOTCH] type
 3 "slope":  str [12, 24] slope
 4 "reso":   linf [0, 10, 101] reso
 5 "drv":    linf [0, 10, 101] drive
 6 "env":    linf [-100, 100, 201] %, env
 7 "att":    logf [10, 250, 101] ms, attack
 8 "hld":    logf [1, 500, 101] ms, hold
 0 "rel":    logf [1, 500, 101] ms, release
 1 "mix":    linf [0, 10, 101] %, mix
 2 "lfo":    linf [linf [0, 10, 101] %, lfo
 6 "spd":    logf [.05, 20, 301] Hz, speed
 7 "phase":  int [0…180] phase
 8 "wave":   str [TRI, SIN, SAW+, SAW-,
                  RMP, SQU, RND] lfo wave
```

Bodyrez

```
 0 "mdl":    BODY
 1 "body":   linf [0, 100, 101] body
```

Rack Amp

```
 0 "mdl":    RACKAMP
 1 "pre":    linf [0, 10, 101] preamp
 2 "buzz":   linf [0, 10, 101] buzz
 3 "punch":  linf [0, 10, 101] punch
 4 "crunch": linf [0, 10, 101] crunch
 5 "drive    linf [0, 10, 101] drive
 6 "out":    linf [0, 10, 101] out gain
 7 "leq":    linf [0, 10, 101] low eq
 8 "heq"     linf [0, 10, 101] high eq
 9 "cab":    int [0, 1] cab sim
```

UK Rock Amp

```
 0 "mdl":    UKROCK
 1 "gain":   linf [0, 10, 101] gains
 2 "bass":   linf [0, 10, 101] bass
 3 "mid":    linf [0, 10, 101] middle
 4 "treb":   linf [0, 10, 101] trebble
 5 "pres     linf [0, 10, 101] presence
 6 "mstr":   linf [0, 10, 101] master
 7 "out":    linf [0, 10, 101] out gain
 8 "sag"     linf [0, 10, 101] sag
 9 "cab":    int [0, 1] cab sim
```

Angel Amp

```
 0 "mdl":    ANGEL
 1 "gain":   linf [0, 10, 101] gains
 2 "bass":   linf [0, 10, 101] bass
 3 "mid":    linf [0, 10, 101] middle
 4 "treb":   linf [0, 10, 101] trebble
 5 "pres     linf [0, 10, 101] presence
 6 "mstr":   linf [0, 10, 101] master
 7 "out":    linf [0, 10, 101] out gain
 8 "sag"     linf [0, 10, 101] sag
 9 "cab":    int [0, 1] cab sim
10 "midb":   int [0, 1] mid boost
11 "bri":    int [0, 1] bright
12 "bt":     int [0, 1] bottom
```

Jazz Clean Amp

```
 0 "mdl":   JAZZC
 1 "vol":   linf [0, 10, 101] volume
 2 "bass":  linf [0, 10, 101] bass
 3 "mid":   linf [0, 10, 101] middle
 4 "treb":  linf [0, 10, 101] trebble
 5 "out":   linf [0, 10, 101] out gain
 6 "bri":   int [0, 1] bright
 7 "cab":   int [0, 1] cab sim
```

Deluxe Amp

```
 0 "mdl":   DELUXE
 1 "vol":   linf [1, 10, 91] volume
 2 "bass":  linf [1, 10, 91] bass
 4 "treb":  linf [1, 10, 91] trebble
 5 "out":   linf [1, 10, 91] out gain
 6 "sag":   linf [1, 10, 91] sag
 7 "cab":   int [0, 1] cab sim
```

Soul Analogue

```
 0 "mdl":   SOUL
 1 "mix":   linf [0, 125, 126] %, mix
 2 "lf":    linf [0, 10, 101] lo freq
 3 "lg":    linf [-5, 5, 101] lo gain
 4 "lmf":   linf [0, 10, 101] lm freq
 5 "lmf3":  int [0, 1] lm /3
 6 "lmq":   linf [0, 10, 101] lm q
 7 "lmg":   linf [-5, 5, 101] lm gain
 8 "hmf":   linf [0, 10, 101] hm freq
 9 "hmf3":  int [0, 1] hm x3
10 "hmq":   linf [0, 10, 101] hm q
11 "hmg":   linf [-5, 5, 101] hm gain
12 "hf":    linf [0, 10, 101] hf freq
13 "hg":    linf [-5, 5, 101] hf gain
```

Even 88 Formant

```
 0 "mdl":   E88
 1 "mix":   linf [0, 125, 126] %, mix
 2 "lf":    linf [0, 10, 101] lf freq
 3 "lg":    linf [-5, 5, 101] lf gain
 4 "lq":    str [LOW, HIGH] lf q
 5 "lt":    str [BELL, SHELV] lf type
 6 "lmf":   linf [0, 10, 101] lm freq
 7 "lmg":   linf [-5, 5, 101] lm gain
 8 "lmq":   linf [0, 10, 101] lm q
 9 "hmf":   linf [0, 10, 101] hm freq
10 "hmg":   linf [-5, 5, 101] hm gain
11 "hmq":   linf [0, 10, 101] hm q
12 "hf":    linf [0, 10, 101] hm freq
13 "hg":    linf [-5, 5, 101] hf gain
14 "hq":    str [LOW, HIG] hf q
15 "ht":    str [BELL, SHELV] hf type
```

Even 84

```
 0 "mdl":   E84
 1 "mix":   linf [0, 125, 126] %, mix
 2 "g":     linf [-20, 20, 81] dB, gain
 3 "lf":    str [OFF, 35, 60, 110, 220] lf freq
 4 "lg":    linf [-5, 5, 101] lf gain
 5 "mf":    str [OFF, 350, 700, 1k6, 3k2,
                 4k8, 7k2] mid freq
 6 "mg":    linf [-5, 5, 101] mid gain
 7 "mq":    str [LOW, HIGH] mid q
 8 "hf":    str [10k, 12k, 16k, OFF] hf freq
 9 "hg":    linf [-5, 5, 101] hf gain
```

Fortissimo110

```
 0 "mdl":   F110
 1 "mix":   linf [0, 125, 126] %, mix
 2 "peq":   int [0, 1] peq on
 3 "Lmf":   linf [0, 10, 101] lm freq
 4 "Lmg":   linf [-5, 5, 101] lm gain
 5 "Lmq":   linf [0, 10, 101] lm q
 6 "Lmf3":  int [0, 1] lm /3
 7 "hmf":   linf [0, 10, 101] hm freq
 8 "hmg":   linf [-5, 5, 101] hm gain
 9 "hmq":   linf [0, 10, 101] hm q
10 "hmf3":  int [0, 1] hm x3
11 "shv":   inf [0, 1] shv on
12 "Lf":    str [33, 56, 95, 160,
                 270, 460] lf freq
13 "lg":    linf [-5, 5, 101] lf gain
14 "hf":    str [3k3, 4k7, 6k8, 10k,
                 15k, 18k] hf freq
15 "hg":    linf [-5, 5, 101] hf q
16 "g":     linf [-18, 18, 73] gain
```

Pulsar

```
 0 "mdl":   PULSAR
 1 "mix":   linf [0, 125, 126] %, mix
 2 "eq1":   int [0, 1] eq1 on
 3 "1lb":   linf [0, 10, 101] lf boost
 4 "1latt": linf [0, 10, 101] lf att
 5 "1lf":   str [20, 30, 60, 100] Hz, lf freq
 6 "1hw":   linf [0, 10, 101] hf wid
 7 "1hb":   linf [0, 10, 101] hf boost
 8 "1hf":   str [3k, 4k, 5k, 8k, 10k,
                 12k, 16k] Hz, hf freq
 9 "1hatt": linf [0, 10, 101] hf att
10 "1hattf":str [5k, 10k, 20k] hf att
11 "eq5":   inf [0, 1] eq5 on
12 "5lb":   linf [0, 10, 101] lm boost
13 "5lf":   str [200, 300, 500, 700,
                 1k] Hz, lf freq
14 "5md":   linf [0, 10, 101] mid dip
15 "5mf":   str [200, 300, 500, 700, 1k, 1k5,
                 2k, 3k, 4k, 5k, 7k] Hz, mid freq
16 "5hb":   linf [0, 10, 101] HM boost
17 "5hf":   str [1k5, 2k, 3k, 4k,
                 5k] Hz, hf freq
```

Mach EQ4

```
 0 "mdl":   MACH4
 1 "mix":   linf [0, 125, 126] %, mix
 2 "sub":   linf [-5, 5, 101] sub
 3 "40":    linf [-5, 5, 101] 40
 4 "160":   linf [-5, 5, 101] 160
 5 "650":   linf [-5, 5, 101] 650
 6 "2k5":   linf [-5, 5, 101] 2k5
 7 "air":   linf [0, 10, 101] air
 8 "airm":  str [OFF, 2k5, 5k, 10k,
                 20k, 40k] air mode
 9 "again": int [0, 1] auto
```

Premium effects

	None *0 **"mdl"**: NONE*
	*External* *0 **"mdl"**: EXT* *1 **"egrp"**: str [OFF, LCL, AUX, A, B, C, SC, USB, CRD, MOD, PLAY, AES] ext grp* *2 **"ein"**: int [1…64] ext in* *3 **"emode"**: str [M, ST, M/S] ext mode* *4 **"lat"**: int [0…200] latency* *5 **"trim"**: linf [-18, 18, 361] dB, trim*
	Hall Reverb *0 **"mdl"**: HALL* *1 **"pdel"**: int [0…200] ms, pre-delay* *2 **"size"**: int [0…100] hall size* *3 **"dcy"**: logf [.2, 5, 101] s, decay* *4 **"mult"**: logf [.5, , 101] bass multiplier* *5 **"damp"**: logf [1k, 20k, 51] Hz, damping* *6 **"lc"**: logf [20, 400, 51] Hz, low cut* *7 **"hc"**: logf [200, 20k, 51] Hz, high cut* *8 **"shp"**: linf [0, 50, 51] shape* *9 **"sprd"**: int [0…50] spread* *10 **"diff"**: int [1…30] diffusion* *11 **"mspd"**: int [0…100] mod speed*
	Room Reverb *0 **"mdl"**: R-OOM* *1 **"pdel"**: int [0…200] ms, pre-delay* *2 **"size"**: linf [4, 76, 145] m, room size* *3 **"dcy"**: logf [.3, 25, 101] s, decay* *4 **"mult"**: logf [.25, 4, 101] bass multiplier* *5 **"damp"**: logf [1k, 20k, 51] Hz, damping* *6 **"lc"**: logf [20, 400, 51] Hz, low cut* *7 **"hc"**: logf [200, 20k, 51] Hz, high cut* *8 **"shp"**: linf [0, 250, 51] shape* *9 **"sprd"**: int [0…50] spread* *10 **"diff"**: int [0…100] diffusion* *11 **"spin"**: int [0…100] spin* *12 **"ecl"**: linf [0, 1200, 1201] ms, echo left* *13 **"ecr"**: linf [0, 1200, 1201] ms, echo right* *14 **"efl"**: linf [-100, 100, 201] %, feed left* *15 **"efr"**: linf [-100, 100, 201] %, feed right*
	Chamber Reverb *0 **"mdl"**: CHAMBER* *1 **"pdel"**: int [0…200] ms, pre-delay* *2 **"size"**: linf [4, 76, 145] m, room size* *3 **"dcy"**: logf [.3, 25, 101] s, decay* *4 **"mult"**: logf [.25, 4, 101] bass multiplier* *5 **"damp"**: logf [1k, 20k, 51] Hz, damping* *6 **"lc"**: logf [20, 400, 51] Hz, low cut* *7 **"hc"**: logf [200, 20k, 51] Hz, high cut* *8 **"shp"**: linf [0, 250, 51] shape* *9 **"sprd"**: int [0…50] spread* *10 **"diff"**: int [0…100] diffusion* *11 **"spin"**: int [0…100] spin* *12 **"ecl"**: linf [0, 300, 301] ms, echo left* *13 **"ecr"**: linf [0, 300, 301] ms, echo right* *14 **"ell"**: fader lvl dB, echo left* *15 **"elr"**: fader lvl dB, echo right*

Plate Reverb

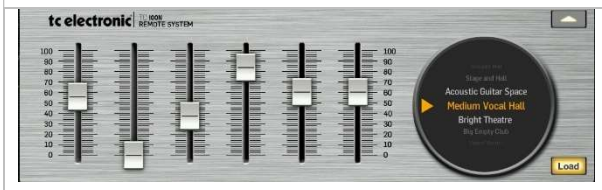
```
 0 "mdl":  PLATE
 1 "pdel": int [0…200] ms, pre-delay
 2 "size": linf [4, 76, 145] m, room size
 3 "dcy":  logf [.3, 25, 101] s, decay
 4 "mult": logf [.25, 4, 101] bass multiplier
 5 "damp": logf [1k, 20k, 51] Hz, damping
 6 "lc":   logf [20, 400, 51] Hz, low cut
 7 "hc":   logf [200, 20k, 51] Hz, high cut
 8 "att":  linf [0, 100, 101] attack
 9 "sprd": int [0…50] spread
10 "diff": int [0…100] diffusion
11 "spin": int [0…100] spin
12 "ecl":  linf [0, 1200, 1201] ms, echo left
13 "ecr":  linf [0, 1200, 1201] ms, echo right
14 "efl":  linf [-100, 100, 201] %, feed left
15 "efr":  linf [-100, 100, 201] %, feed right
```

Concert Reverb

```
 0 "mdl":  CONCERT
 1 "pdel": int [0…200] ms, pre-delay
 2 "size": linf [20, 76, 113] m, room size
 3 "dcy":  logf [.3, 29, 51] s, decay
 4 "mult": logf [.25, 4, 101] bass multiplier
 5 "damp": logf [1k, 20k, 51] Hz, damping
 6 "lc":   logf [20, 400, 51] Hz, low cut
 7 "hc":   logf [200, 20k, 51] Hz, high cut
 8 "shp":  linf [0, 50, 51] shape
 9 "sprd": int [0…50] spread
10 "diff": int [1…16] diffusion
11 "depth":int [0, 100] depth
12 "rfl":  linf [0, 1200, 1201] ms, refl. left
13 "rfr":  linf [0, 1200, 1201] ms, refl. right
14 "rfll": fader lvl dB, reflection left
15 "rflr": fader lvl dB, reflection right
16 "spin": int [0…100] spin
17 "crs":  int [1…100] chorus
```

Ambiance

```
 0 "mdl":  AMBI
 1 "pdel": int [0…200] ms, pre-delay
 2 "size": linf [2, 100, 99] m, room size
 3 "dcy":  logf [.2, 7.3, 101] s, decay
 4 "tail": int [0…100] tail gain
 5 "damp": logf [1k, 20k, 51] Hz, damping
 6 "diff": int [1…30] diffusion
 7 "mod":  int [1…100] modulation speed
 8 "lc":   logf [20, 400, 51] Hz, low cut
 9 "hc":   logf [200, 20k, 51] Hz, high cut
```



VSS3 Reverb

```
 0 "mdl":    VSS3
 1 "preset": str [Build, Small Booth, Home Room,
                  Dialog Alley, Small Wood Room,
                  A Small Room, Intimate Studio,
                  Small Room, Tight & Natural, Room
                  Conversation, Furnished Room 2,
                  Studio 20x20 ft, Drew Room, Piano
                  Close, Clear Guitar Room, Wide
                  Ambient Chamber, Small Dense Hall,
                  Slap Hall, Acoustic Gtr Ambience,
                  Clear Room, Livingroom, Band
                  Rehearsal Room, The Studio, In The
                  Room, Studio 40x40 ft, Hit Room,
                  Ambient Hall, Stage and Hall,
                  Acoustic Guitar Space, Medium
                  Vocal Hall, Bright Theatre , Big
                  Empty Club, Venue Warm 1, Concert
                  1, Bright Guitar Hall, Concert
                  Arena, Concert Piano, Piano Hall
                  1st Row, Empty Arena, Ballad Vocal
```

	Hall, Grand Vocal Hall, Large Warm Hall, Back There, WoodHall, Church, Sound Col, 5000 Hall, Cathedral, Large Church, Medium Church, Warm Cathedral, Cologne Cathedral, Drum Plate Stuff, Drum Wood Plate, Piano Plate, Stairway Plate, Slapback Plate, Ambient Plate, Silky Gold Plate, Gold Plate, EMT 141, Leader Of The Band, VocalDry, Vocal Room, VocalWet, Slapback Vox 1, Vocal Hall 1, Vocal Chamber, Bright Male Vox, Vocal Bright, Vocal Deep, Vocal Female, Vocal Deep Male, Large Vocal Hall, Kick & Bass Ambience, Drum Room, Small Perc Room, Drum Room Xpander, Bright Shoe Gaze Snare, Snare Room Bright, Tom-Tom Reverb, Bossa Nova Perc Room, Hard Drum Space, Puk Drum Ambience, Overhead Mics, Dance Snare, Drum Perc Soft 1, Perc Straight Tail, Store Room, Studio Small, The Alley, Near The Wall, WoodFlr, Large Office, Conference Room, Dance Studio, Forest 2, StoneWall, Venue 1, Small Stairway, Forest 1, Airport PA, Small Tower Hall, On The Street, Dark Tunnel, Empty Nightclub, Parking Garage, Parking Distant, Long Swimmingpool] preset 2 **"load"**: int [0, 1] load 3 **"erpdly"**: linf [0, 100, 101] ms, er pdly 4 **"ertype"**: str [ROYAL, THEATRE, CHURCH, GAS, CONCERT, ROYAL2, V1-NEAR, V2-HARD, V3-SPREAD, V4-BUILD, V5-RANDOM, SLAP, CAR, PHONEBTH, BATHROOM, CONFRM9, CONFRM30, GARAGE, SWIMSTDM, AIRPORT, STREET, ALLEY, PIAZA, FOREST], er type 5 **"ersize"**: str [SML, MED, LRG] er size 6 **"erpos"**: str [NEAR, DIST] er position 7 **"erbal"**: linf [-100, 100, 201], er balance 8 **"erlc"**: logf [20, 400, 51] Hz, er low cut 9 **"ercol"**: linf [-40, 40, 81] er color 10 **"erlvl"**: fader lvl dB, er level 11 **"rvtype"**: str [SMOOTH, NATURAL, ALIVE, FAST, X-WIDE, ALIVE2] rev type 12 **"rvwide"**: str [NARROW, NORMAL, WIDE, X-WIDE] rev wide 13 **"rvpdly"**: linf [0, 200, 201] ms, rev pdly 14 **"dcy"**: linf [.1, 20, 280] s, decay 15 **"diff"**: linf [-50, 50, 101] diffuse 16 **"rvbal"**: linf [-100, 100, 201] balance 17 **"rvlvl"**: fader lvl dB, reverb level 18 **"ldcy"**: linf [.1, 2.5, 25] low decay 19 **"lmdcy"**: linf [.1, 2.5, 25] lowmid decay 20 **"hmdcy"**: linf [.1, 2.5, 25] mid decay 21 **"hdcy"**: linf [.1, 2.5, 25] high decay 22 **"hsoft"**: linf [-50, 50, 101] high soft 23 **"lxo"**: logf [20, 500, 113] Hz, low xover 24 **"mxo"**: logf [200, 2000, 81] Hz, mid xover 25 **"hxo"**: logf [500, 20000, 105] Hz, high xover 26 **"lshv"**: logf [20, 200, 81] Hz, low shelf 27 **"lsdmp"**: linf [0, -18, 37] dB, low damp 28 **"hcut"**: logf [20, 20000, 241] Hz, high cut 29 **"mtype"**: str [A, B, C, D, E, F] modulation type 30 **"mrate"**: linf [-100, 100, 201] modulation rate 31 **"mwid"**: linf [0, 200, 201] modulation width 32 **"view"**: int [0, 1] view

Vintage Room

```
 0 "mdl":   V-ROOM
 1 "pdel":  int [0…200] ms, pre-delay
 2 "size":  int [0…50] size
 3 "dcy":   logf [.1, 20, 101] s, decay
 4 "dens":  linf [1, 30, 30] density
 5 "erlvl": linf [0, 100, 101] %, Early level
 6 "lmult": logf [.1, 10, 101] low multiplier
 7 "hmult": logf [.1, 10, 101] high multiplier
 8 "lc":    logf [20, 400, 51] Hz, low cut
 9 "hc":    logf [200, 20k, 51] Hz, high cut
10 "frz":   int [0, 1] freeze
```

Vintage Reverb,

```
 0 "mdl":   V-REV
 1 "pdel":  int [0…120] ms, pre-delay
 2 "dcy":   linf [.4, 4.5, 83] s, decay
 3 "lmult": logf [.5, 2, 51] low multiplier
 4 "hmult": logf [.25, .67, 51] high multiplier
 5 "mod":   int [0…100] modulation speed
 6 "lc":    logf [20, 400, 51] Hz, low cut
 7 "hc":    logf [5000, 20k, 31] Hz, high cut
 8 "out":   str [FRONT, REAR] output
 9 "trans": int [0…1] transformer
```

Vintage Plate

```
 0 "mdl":   V-PLATE
 1 "pdel":  int [0…250] ms, pre-delay
 2 "dcy":   linf [1, 6, 101] s, decay
 3 "lc":    logf [20, 400, 51] Hz, low cut
 4 "col":   linf [-20, 20, 42] color
```

Blue Plate

```
 0 "mdl":   BPLATE
 1 "pdel":  int [0…200] ms, pre delay
 2 "size":  int [0…100] ms, size
 3 "dcy":   logf [0.2, 5, 101] s, decay
 4 "mult":  logf [0.5, 2, 51] bass multiplier
 5 "damp":  logf [1000, 20000, 51] Hz, damping
 6 "lc":    logf [20, 400, 51] Hz, low cut
 7 "hc":    logf [200, 20000, 51] Hz, high cut
 8 "xover": logf [20, 500, 51] Hz, xover
 9 "mdep":  linf [1, 50, 50] modulation depth
10 "msdp":  int [0…100] modulation speed
11 "diff":  int [1…30] diffusion
```

Gated Reverb

```
 0 "mdl":  GATED
 1 "pdel": int [0…200] ms, pre-delay
 2 "att":  int [4…30] attack
 3 "dcy":  logf [.14, 1, 101] s, decay
 4 "dens": int [0…100] density
 5 "diff": int [0…100] diffusion
 6 "sprd": int [0…50] spread
 7 "lc":   logf [20, 400, 51] Hz, low cut
 8 "hfs":  logf [200, 20k, 51] Hz, high freq
 9 "hsg":  linf [-30, 0, 61] dB, high gain
```

Reverse Reverb

```
 0 "mdl":  REVERSE
 1 "pdel": int [0…200] ms, pre-delay
 2 "rise": int [4…50] rise
 3 "dcy":  logf [.14, 1, 101] s, decay
 4 "diff": int [0…30] diffusion
 5 "sprd": int [0…100] spread
 6 "lc":   logf [20, 400, 51] Hz, low cut
 7 "hfs":  logf [200, 20k, 51] Hz, high freq
```

	8 “hsg”: linf [-30, 0, 61] dB, high gain
	Delay/Reverb *0 “mdl”: DEL/REV* *1 “time”: linf [0, 3000, 3000] ms, time* *2 “feed”: linf [0, 100, 101] %, feed* *3 “fhc”: logf [200, 2000, 51] Hz, feed HC* *4 “dly”: linf [0, 100, 101] %, delay* *5 “d2r”: linf [0, 100, 101] %, delay→rev* *6 “pdel”: int [0…200] ms, pre delay* *7 “size”: int [2…100] size* *8 “dcy”: logf [.1, 5, 51] s, decay* *9 “damp”: logf [1000, 20k, 51] Hz, damp* *10 “rlc”: logf [20, 400, 51] Hz, rev LC* *11 “i2r”: linf [0, 100, 101] %, in→rev*
	Shimmer Reverb *0 “mdl”: SHIMMER* *1 “pdel”: int [0…250] ms, pre delay* *2 “size”: int [2…50] size* *3 “dcy”: logf [1, 20, 101] s, decay* *4 “lc”: logf [25, 250, 51] Hz, low cut* *5 “hc”: logf [500, 7000, 51] Hz, high cut* *6 “damp”: linf [0, 100, 101] %, damp* *7 “shim”: linf [0, 100, 101] %, shimmer* *8 “shine”: linf [0, 100, 101] %, shine*
	Spring Reverb *0 “mdl”: SPRING* *1 “dcy”: logf [1.5, 6, 101] s, decay* *2 “dens”: linf [1, 30, 30] density* *3 “low”: linf [1, 50, 50] bass* *4 “high”: linf [1, 50, 50] trebble*
	Dimension CRS *0 “mdl”: DIMCRS* *1 “sw1”: int [0, 1] sw1* *2 “sw2”: int [0, 1] sw2* *3 “sw3”: int [0, 1] sw3* *4 “sw4”: int [0, 1] sw4* *5 “in”: str [MONO, STEREO] input* *6 “drysw”: int [0, 1] dry*
	Stereo Chorus *0 “mdl”: CHORUS* *1 “lc”: logf [20, 400, 51] Hz, LC* *2 “hc”: logf [200, 20000, 51] Hz, HC* *3 “wave”: linf [0, 100, 101] waveform* *4 “phase”: linf [0, 100, 101] phase* *5 “mix”: linf [0, 100, 101] %, mix* *6 “dlyl”: linf [5, 50, 226] ms, dely l* *7 “dlyr”: linf [5, 50, 226] ms, dely r* *8 “depl”: linf [0, 100, 101] %, depth l* *9 “depr”: linf [0, 100, 101] %, depth r* *10 “sprd”: linf [0, 100, 101] %, spread* *11 “spd”: logf [.05, 5, 201] Hz, speed*
	Stereo Flanger *0 “mdl”: CHORUS* *1 “lc”: logf [20, 400, 51] Hz, LC* *2 “hc”: logf [200, 20000, 51] Hz, HC* *3 “flc”: logf [20, 400, 51] Hz, feed LC* *4 “fhc”: logf [200, 20000, 51] Hz, feed HC* *5 “mix”: linf [0, 100, 101] %, mix* *6 “dlyl”: linf [5, 20, 196] ms, dely l* *7 “dlyr”: linf [5, 20, 196] ms, dely r* *8 “depl”: linf [0, 100, 101] %, depth l* *9 “depr”: linf [0, 100, 101] %, depth r*

	*10 “**phase**”: linf [0, 180, 181] phase* *11 “**spd**”: logf [.05, 5, 201] Hz, speed* *12 “**feed**”: linf [-90, 90, 181] %, feed*
	Stereo Delay *0 “**mdl**”: ST-DL* *1 “**time**”: linf [1, 3000, 3000] ms, time* *2 “**mode**”: str [ST, X, M] mode* *3 “**fact**”: str [1/4, 1/3, 1/2, 2/3, 3/4, 1, 3/8, 5/4, 4/3, 3/2, 2] factor* *4 “**pat**”: str [1/2:1, 2/3:1, 3/4:1, 7/8:1, 1:1, 1:9/8, 1:5/4, 1:4/3, 1:3/2] pattern* *5 “**offset**”:int [-50…50] ms, offset* *6 “**feed**”: linf [0, 100, 101] %, feed* *7 “**flc**”: logf [20, 400, 51] Hz, feed L cut* *8 “**fhc**”: logf [200, 20000, 51] Hz, feed H cut* *9 “**lc**”: logf [20, 400, 51] Hz, low cut* *10 “**hc**”: logf [200, 20000, 51] Hz, high cut*
	UltraTap Delay *0 “**mdl**”: TAP-DL* *1 “**time**”: linf [1, 2000, 2000] ms, time* *2 “**rep**”: int [1..16] repeat* *3 “**slp**”: linf [-6, 6, 121] dB, slope* *4 “**fact**”: str [1/3, 1/2, 2/3, 3/4, 1, 5/4, 4/3, 3/2, 2] factor* *5 “**pdel**”: linf [0, 500, 501] ms, pre delay* *6 “**mode**”: str [MOVE, JUMP, FOCUS, SPREAD] mode* *7 “**wid**”: linf [-100, 100, 201] %, width* *8 “**diff**”: linf [0, 100, 101] diffusion* *9 “**lc**”: logf [20, 400, 51] Hz, low cut* *10 “**hc**”: logf [200, 20000, 51] Hz, high cut*
	Tape Delay *0 “**mdl**”: TAPE-DL* *1 “**time**”: linf [60, 650, 591] ms, time* *2 “**sust**”: linf [0, 100, 101] %, sustain* *3 “**drv**”: linf [0, 100, 101] %, drive* *4 “**wf**”: linf [0, 100, 101] %, flutter*
	OilCan Delay *0 “**mdl**”: OILCAN* *1 “**time**”: linf [0, 10, 1001] time* *2 “**sust**”: linf [0, 10, 101] %, sustain* *3 “**wb**”: linf [0, 10, 101] %, wobble* *4 “**tone**”: linf [0, 10, 101] %, tone*
	BBD Delay *0 “**mdl**”: BBD-DL* *1 “**dly**”: linf [0, 100, 1001] time* *2 “**feed**”: linf [0, 100, 101] %, feed*
	Stereo Pitch *0 “**mdl**”: PITCH* *1 “**semi**”: int [-12…12] semitones* *2 “**cent**”: int [-50…50] cent* *3 “**dly**”: linf [0, 500, 501] ms, delay* *4 “**lc**”: logf [20, 400, 51] Hz, low cut* *5 “**hc**”: logf [200, 20000, 51] Hz, high cut* *6 “**mix**”: linf [0, 100, 101] %, mix*

Dual Pitch

```
 0 "mdl":   D-PITCH
 1 "semi1": int [-12…12] semitones 1
 2 "cent1": int [-50…50] cent 1
 3 "dly1":  linf [0, 500, 501] ms, delay 1
 4 "pan1":  linf [-100, 100, 201] %, pan 1
 5 "lvl1":  fader lvl 1 dB
 6 "semi2": int [-12…12] semitones 2
 7 "cent2": int [-50…50] cent 2
 8 "dly2":  linf [0, 500, 501] ms, delay 2
 9 "pan2":  linf [-100, 100, 201] %, pan 2
10 "lvl2":  fader lvl 2 dB
11 "lc":    logf [20, 400, 51] Hz, low cut
12 "hc":    logf [200, 20000, 51] Hz, high cut
```

Channel effects

```
None
 0 “mdl”:    NONE
```


```
External
 0 “mdl”:    EXT
 1 “egrp”:   str [OFF, LCL, AUX, A, B, C, SC,
                 USB, CRD, MOD, PLAY, AES] ext grp
 2 “ein”:    int [1…64] ext in
 3 “emode”:  str [M, ST, M/S] ext mode
 4 “lat”:    int [0…200] latency
 5 “trim”:   linf [-18, 18, 361] dB, trim
```

Soul Analog EQ

```
 0 “mdl”:    SOUL
 1 “mix”:    linf [0, 125, 126] %, mix
 2 “lf”:     linf [0, 10, 101] lo freq
 3 “lg”:     linf [-5, 5, 101] lo gain
 4 “lmf”:    linf [0, 10, 101] lm freq
 5 “lmf3”:   int [0, 1] lm /3
 6 “lmq”:    linf [0, 10, 101] lm q
 7 “lmg”:    linf [-5, 5, 101] lm gain
 8 “hmf”:    linf [0, 10, 101] hm freq
 9 “hmf3”:   int [0, 1] hm x3
10 “hmq”:    linf [0, 10, 101] hm q
11 “hmg”:    linf [-5, 5, 101] hm gain
12 “hf”:     linf [0, 10, 101] hf freq
13 “hg”:     linf [-5, 5, 101] hf gain
```

Even 88-Formant EQ

```
 0 “mdl”:    E88
 1 “mix”:    linf [0, 125, 126] %, mix
 2 “lf”:     linf [0, 10, 101] lf freq
 3 “lg”:     linf [-5, 5, 101] lf gain
 4 “lq”:     str [LOW, HIGH] lf q
 5 “lt”:     str [BELL, SHELV] lf type
 6 “lmf”:    linf [0, 10, 101] lm freq
 7 “lmg”:    linf [-5, 5, 101] lm gain
 8 “lmq”:    linf [0, 10, 101] lm q
 9 “hmf”:    linf [0, 10, 101] hm freq
10 “hmg”:    linf [-5, 5, 101] hm gain
11 “hmq”:    linf [0, 10, 101] hm q
12 “hf”:     linf [0, 10, 101] hm freq
13 “hg”:     linf [-5, 5, 101] hf gain
14 “hq”:     str [LOW, HIG] hf q
15 “ht”:     str [BELL, SHELV] hf type
```

Even 84 EQ

```
 0 “mdl”:    E84
 1 “mix”:    linf [0, 125, 126] %, mix
 2 “g”:      linf [-20, 20, 81] dB, gain
 3 “lf”:     str [OFF, 35, 60, 110, 220] lf freq
 4 “lg”:     linf [-5, 5, 101] lf gain
 5 “mf”:     str [OFF, 350, 700, 1k6, 3k2,
                 4k8, 7k2] mid freq
 6 “mg”:     linf [-5, 5, 101] mid gain
 7 “mq”:     str [LOW, HIGH] mid q
 8 “hf”:     str [10k, 12k, 16k, OFF] hf freq
 9 “hg”:     linf [-5, 5, 101] hf gain
```

Focusrite ISA 110 EQ

```
 0 “mdl”:    F110
 1 “mix”:    linf [0, 125, 126] %, mix
 2 “peq”:    int [0, 1] peq on
 3 “lmf”:    linf [0, 10, 101] lm freq
 4 “lmg”:    linf [-5, 5, 101] lm gain
 5 “lmq”:    linf [0, 10, 101] lm q
```

	6 **"Lmf3"**: int [0, 1] Lm /3 7 **"hmf"**: Linf [0, 10, 101] hm freq 8 **"hmg"**: Linf [-5, 5, 101] hm gain 9 **"hmq"**: Linf [0, 10, 101] hm q 10 **"hmf3"**: int [0, 1] hm x3 11 **"shv"**: inf [0, 1] shv on 12 **"Lf"**: str [33, 56, 95, 160, 270, 460] Lf freq 13 **"Lg"**: Linf [-5, 5, 101] Lf gain 14 **"hf"**: str [3k3, 4k7, 6k8, 10k, 15k, 18k] hf freq 15 **"hg"**: Linf [-5, 5, 101] hf q 16 **"g"**: Linf [-18, 18, 73] gain
	Pulsar P1a/M5 EQ 0 **"mdl"**: PULSAR 1 **"mix"**: Linf [0, 125, 126] %, mix 2 **"eq1"**: int [0, 1] eq1 on 3 **"1lb"**: Linf [0, 10, 101] Lf boost 4 **"1latt"**: Linf [0, 10, 101] Lf att 5 **"1lf"**: str [20, 30, 60, 100] Hz, Lf freq 6 **"1hw"**: Linf [0, 10, 101] hf wid 7 **"1hb"**: Linf [0, 10, 101] hf boost 8 **"1hf"**: str [3k, 4k, , 5k, 8k, 10k, 12k, 16k] Hz, hf freq 9 **"1hatt"**: Linf [0, 10, 101] hf att 10 **"1hattf"**:str [5k, 10k, 20k] hf att 11 **"eq5"**: int [0, 1] eq5 on 12 **"5lb"**: Linf [0, 10, 101] Lm boost 13 **"5lf"**: str [200, 300, 500, 700, 1k] Hz, Lf freq 14 **"5md"**: Linf [0, 10, 101] mid dip 15 **"5mf"**: str [200, 300, 500, 700, 1k, 1k5, 2k, 3k, 4k, 5k, 7k] Hz, mid freq 16 **"5hb"**: Linf [0, 10, 101] HM boost 17 **"5hf"**: str [1k5, 2k, 3k, 4k, 5k] Hz, hf freq
	Mach EQ4 0 **"mdl"**: MACH4 1 **"mix"**: Linf [0, 125, 126] %, mix 2 **"sub"**: Linf [-5, 5, 101] sub 3 **"40"**: Linf [-5, 5, 101] 40 4 **"160"**: Linf [-5, 5, 101] 160 5 **"650"**: Linf [-5, 5, 101] 650 6 **"2k5"**: Linf [-5, 5, 101] 2k5 7 **"air"**: Linf [0, 10, 101] air 8 **"airm"**: str [OFF, 2k5, 5k, 10k, 20k, 40k] air mode 9 **"again"**: int [0, 1] auto
	Even Channel Even 88 Gate, Even 88 Formant EQ, Even Compressor/Limiter 0 **"mdl"**: *EVEN* 1 **"g_thr"**: Linf [-40.0, 0.0, 81] dB 2 **"g_hyst"**: Linf [0.0, 25.0, 51] dB 3 **"g_range"**: Linf [0, 60, 61] dB 4 **"g_rel"**: Logf [100, 3000, 130] ms 5 **"g_fast"**: int [0, 1] 6 **"g_m40"**: int [0, 1] 7 **"g_on"**: int [0, 1] 8 **"eq_on"**: int [0, 1] 9 **"lf"**: Linf [0.0, 10.0, 101] 10 **"lg"**: Linf [-5.0, 5.0, 101] 11 **"lq"**: Str [LOW, HIGH] 12 **"lt"**: Str [BELL, SHELV] 13 **"lmf"**: Linf [0.0, 10.0, 101] 14 **"lmg"**: Linf [-5.0, 5.0, 101] 15 **"lmq"**: Linf [0.0, 10.0, 101] 16 **"hmf"**: Linf [0.0, 10.0, 101]

```
17 "hmg":      Linf [-5.0, 5.0, 101]
18 "hmq":      Linf [0.0, 10.0, 101]
19 "hf":       Linf [0.0, 10.0, 101]
20 "hg":       Linf [-5.0, 5.0, 101]
21 "hq":       Str [LOW, HIGH]
22 "ht":       Str [BELL, SHELV]
23 "mix":      Linf [0, 125 %, 126]
24 "d_on":     int [0, 1]
25 "d_thr":    Linf [-12.0, 0.0, 25] dB
26 "d_rec":    Str [50, 100, 200, 800, A1, A2]
27 "d_fast":   int [0, 1]
28 "d_con":    int [0, 1]
29 "d_cthr":   Linf [-35.0, -5.0 dB, 61]
30 "d_ratio":  Str [1.5, 2.0, 3.0, 4.0, 6.0]
31 "d_crec":   Str [100, 400, 800, 1500, A1, A2]
32 "d_cfast":  int [0, 1]
33 "d_gain":   Linf [-6, 12 dB, 7]
```

Soul Channel

Soul 9000 Gate/Expander, Soul AnaLogue EQ, Soul 9000 Channel Compressor

```
 0 "mdl":      *SOUL*
 1 "g_thr":    Linf [-40.0, 0.0, 81] dB
 2 "g_range":  Linf [0, 40, 41] dB
 3 "g_hld":    Logf [10, 4000, 130] ms
 4 "g_rel":    Logf [100, 4000, 130] ms
 5 "g_fast":   int [0, 1]
 6 "g_mode":   Str [GATE, EXP]
 7 "g_on":     int [0, 1]
 8 "eq_on":    int [0, 1]
 9 "lf":       Linf [0.0, 10.0, 101]
10 "lg":       Linf [-5.0, 5.0, 101]
11 "lmf":      Linf [0.0, 10.0, 101]
12 "lmf3":     int [0, 1]
13 "lmq":      Linf [0.0, 10.0, 101]
14 "lmg":      Linf [-5.0, 5.0, 101]
15 "hmf":      Linf [0.0, 10.0, 101]
16 "hmf3":     int [0, 1]
17 "hmq":      Linf [0.0, 10.0, 101]
18 "hmg":      Linf [-5.0, 5.0, 101]
19 "hf":       Linf [0.0, 10.0, 101]
20 "hg":       Linf [-5.0, 5.0, 101]
21 "mix":      Linf [0, 125 %, 126]
22 "d_on":     int [0, 1]
23 "d_thr":    Linf [-30.0, 18.0, 97] dB
24 "d_ratio":  Str [1.3, 1.4, 1.6, 1.8, 2.0,
                    2.5, 2.8, 3.3, 4.0, 5.0,
                    6.0, 7.0, 9.0, 12, 20,
                    50, 100]
25 "d_fast":   int [0, 1]
26 "d_rel":    Logf [100, 4000, 65] ms
27 "d_peak":   int [0, 1]
```

Vintage Channel

76 Limiting Amplifier, Pulsar EQ P1A/m5, Model 2A Leveling Amplifier

```
 0 "mdl":      *VINTAGE*
 1 "d_in":      Linf [-48.0, 0.0, 97] dB
 2 "d_out":     Linf [-48.0, 0.0, 97] dB
 3 "d_att":     Linf [1.0, 7.0, 61]
 4 "d_rel":     Linf [1.0, 7.0, 61]
 5 "d_ratio":   Str [4, 8, 12, 20, ALL]
 6 "d_on":      int [0, 1]
 7 "eq1":       int [0, 1]
 8 "1lb":       Linf [0.0, 10.0, 101]
 9 "1latt":     Linf [0.0, 10.0, 101]
10 "1lf":       Str [20, 30, 60, 100]
11 "1hw":       Linf [0.0, 10.0, 101]
12 "1hb":       Linf [0.0, 10.0, 101]
13 "1hf":       Str [3k, 4k, 5k, 8k, 10k, 12k, 16k]
14 "1hatt":     Linf [0.0, 10.0, 101]
15 "1hattf":    Str [5k, 10k, 20k]
```

```
16 “eq5”:       int [0, 1]
17 “5lb”:       Linf [0.0, 10.0, 101]
18 “5lf”:       Str [200, 300, 500, 700, 1k]
19 “5md”:       Linf [0.0, 10.0, 101]
20 “5mf”:       Str [200, 300, 500, 700, 1k, 1k5,
                     2k, 3k, 4k, 5k, 7k]
21 “5hb”:       Linf [0.0, 10.0, 101]
22 “5hf”:       Str [1k5, 2k, 3k, 4k, 5k]
23 “l_ingain”: Linf [0, 100, 101]
24 “l_peak”:   Linf [0, 100, 101]
25 “l_mode”:   Str [COMP, LIM]
26 “l_on”:     int [0, 1]
```

Bus Channel

Soul Warmth, Even 84 EQ, Soul G Bus Compressor

```
 0 “mdl”:      *BUS*
 1 “w_drv”:    Linf [10, 125, 116] %
 2 “w_hrm”:    Linf [-100, 100, 201]
 3 “w_col”:    Linf [-1.00, +1.00, 41]
 4 “w_trim”:   Linf [-18.0, +6.0, 49] dB
 5 “w_mix”:    Linf [0, 100, 101] %
 6 “w_on”:     int [0, 1]
 7 “eq_on”:    int [0, 1]
 9 “g”:        Linf [-20.0, 20.0, 81] dB
10 “lf”:       Str [OFF, 35, 60, 110, 220]
11 “lg”:       Linf [-5.0, 5.0, 101]
12 “mf”:       Str [OFF, 350, 700, 1k6, 3k2,
                    4k8, 7k2]
13 “mg”:       Linf [-5.0, 5.0, 101]
14 “mq”:       Str [LOW, HIGH]
15 “hf”:       Str [10k, 12k, 16k, OFF]
16 “hg”:       Linf [-5.0, 5.0, 101]
17 “mix”:      Linf [0, 125 %, 126]
18 “d_thr”:    Linf [-40.0, 0.0, 81] dB
19 “d_ratio”:  Str [1.5, 2.0, 3.0, 4.0, 5.0, 10]
20 “d_att”:    Str [0.1, 0.3, 1.0, 3.0, 10.0, 30.0]
21 “d_rel”:    Str [0.1, 0.2, 0.4, 0.8, 1.6, AUTO]
22 “d_gain”:   Linf [-6.0, 12.0, 37] dB
23 “d_on”:     int [0, 1]
```

Mastering

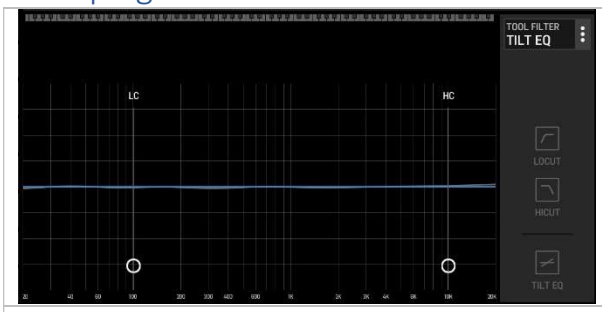
Tape, Mach EQ4 EQ, Stereo Enhancer, Precision Limiter

```
 0 “mdl”:      *MASTER*
 1 “t_drv”:    Linf [-5.0, 25.0, 61] dB
 2 “t_spd”:    Logf [7.5, 30.0, 65]
 3 “t_low”:    int [0, 1]
 4 “t_hi“:     int [0, 1]
 5 “t_on”:     int [0, 1]
 6 “sub”:      Linf [-5.0, 5.0, 201]
 7 “40”:       Linf [-5.0, 5.0, 201]
 8 “160”:      Linf [-5.0, 5.0, 201]
 9 “650”:      Linf [-5.0, 5.0, 201]
10 “2k5”:      Linf [-5.0, 5.0, 201]
11 “air”:      Linf [0.0, 10.0, 201]
12 ”airm”:     Str [OFF, 2k5, 5k, 10k, 20k, 40k]
13 “eq_on”:    int [0, 1]
14 “e_stlvl”: Linf [-100, +100, 201] %
15 “e_lmf”:   Linf [-100, +100, 201] %
16 “e_mlvl”:  Linf [-100, +100, 201] %
17 “e_st”:    Linf [-100, 100, 201] %
18 “e_m”:     Linf [-100, 100, 201] %
19 “e_bass”:  Linf [0, 100, 101] %
20 “e_mid”:   Linf [0, 100, 101] %
21 “e_high”:  Linf [0, 100, 101] %
22 “e_bassf”: Linf [1, 50, 50]
23 “e_midq”:  Linf [1, 50, 50]
24 “e_highf”: Linf [1, 50, 50]
25 “e_on”:    int [0, 1]
26 “l_gin”:   Linf [0.00, 18.00, 73] dB
27 “l_gout”:  Linf [-18.00, 0.00, 73] dB
28 “l_sqz”:   int [0, 100]
```

	29 **“l_knee”:** int [0, 10] 30 **“l_again”:** int [0, 1] 31 **“l_att”:** Linf [0.05, 1.00, 96] ms 32 **“l_rel”:** Logf [20, 2000, 101] ms

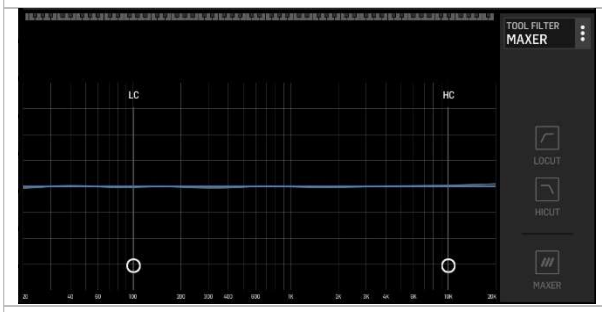
Plugins

Filter plugins



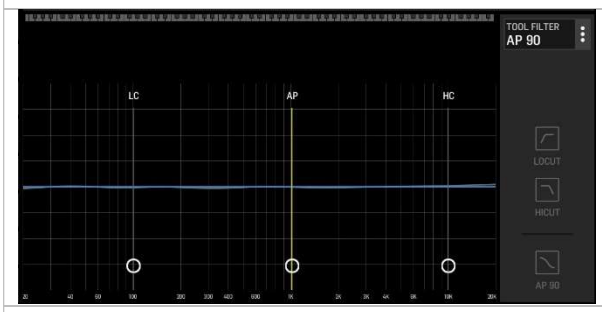
Tilt Filter

*0 “**mdl**”: TILT*
*1 “**tilt**”: linf [-6, 6, 49] tilt*



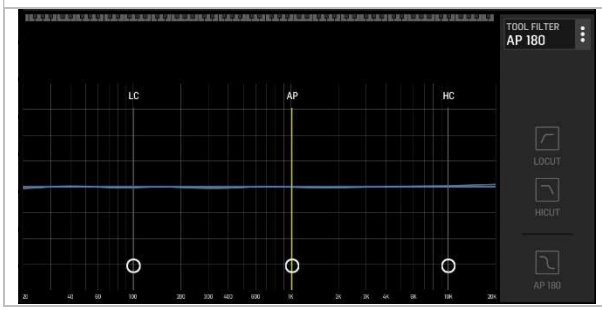
Maxer Filter

*0 “**mdl**”: MAX*
*1 “**low**”: linfplugins [0, 100, 101] %, low cont*
*2 “**proc**”: linf [0, 100, 101] %, high proc*



AP90 Filter (all pass)

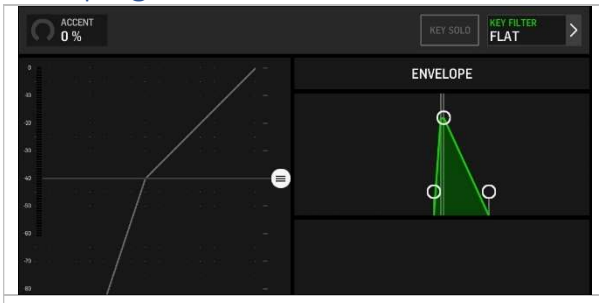
*0 “**mdl**”: AP1*
*1 “**freq**”: logf [100, 10000, 100] Hz, freq*



AP180 Filter (all pass)

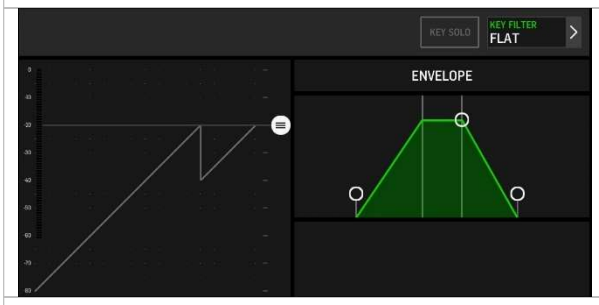
*0 “**mdl**”: AP2*
*1 “**f**”: logf [100, 10000, 100] Hz, freq*
*2 “**q**”: logf [.442, 10, 181] q*

Gate plugins



Standard Gate/Expander

```
0 "mdl":   GATE
1 "thr":   linf [-80, 0, 161] dB, thr
2 "range": linf [3, 60, 115] dB, range
3 "att":   linf [0, 120, 121] ms, attack
4 "hold":  linf [1, 200, 200] ms, hold
5 "rel":   logf [4, 4000, 130] ms, release
6 "acc":   linf [0, 100, 21] %, accent
7 "ratio": str [1:1.5, 1:2, 1:3, 1:4,
                gate] ratio
```



Standard Ducker

```
0 "mdl":   DUCK
1 "thr":   linf [-80, 0, 161] dB, thr
2 "range": linf [3, 60, 115] dB, range
3 "att":   linf [0, 120, 121] ms, attack
4 "hold":  linf [1, 200, 200] ms, hold
5 "rel":   linf [20, 4000, 130] ms, release
```

SSL 9000 Gate/Expander

```
0 "mdl":    9000G
1 "thr":    linf [-40, 0 81] dB, input
2 "range":  linf [-0, 40, 41] dB
3 "hld":    logf [10, 4000, 130] ms, hold
4 "rel":    logf [100, 4000, 130] ms, release
5 "fast":   int [0, 1] fast
6 "mode":   str [GATE, EXP] mode
```

Even 88-Gate

```
0 "mdl":   E88
1 "thr":   linf [-40, 0, 81] dB, thr
2 "hyst":  linf [0, 25, 51] dB, hyst
3 "range": linf [0, 60, 61] dB, range
4 "rel":   logf [100, 3000, 130] ms, release
5 "fast":  int [0, 1] fast
6 "m40":   int [0, 1] thr
```

DrawMore Expander Gate 241

```
0 "mdl":   DUCK
1 "thr":   linf [-80, 0, 161] dB, thr
2 "slow":  int [0, 1] slow
```

DBX 902 De-Esser

```
0 "mdl":   DS902
1 "f":     logf [800, 8000, 130] Hz, freq
2 "range": linf [3, 12, 25] dB, range
3 "mode":  str [FULL, HF] mode
```

76 Limiting Amp

```
0 "mdl":   76LA
1 "in":    linf [-48, 0, 97] dB, input
2 "out":   linf [-48, 0, 97] dB
3 "att":   linf [1, 7, 61] attack
4 "rel":   linf [1, 7, 61] release
5 "ratio": str [4, 8, 12, 20, ALL] ratio
```

Leveling Amplifier 2A

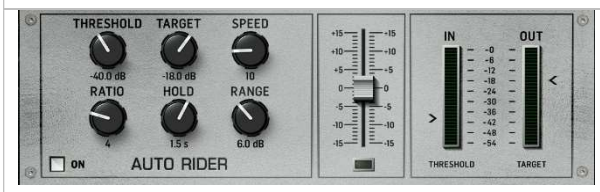
```
0 "mdl":    LA
1 "ingain": linf [0, 100, 101] gain
2 "peak":   linf [0, 100, 101] peak
3 "mode":   str [comp, lim] mode
```

Source Extractor

```
0 "mdl":    PSE
1 "thr":    linf [-36, 12, 97] dB, threahold
2 "depth":  linf [0, 20, 41] dB, depth
3 "fast":   int [0, 1] fast
4 "peak":   int [0, 1] peak
```

Wave Designer

```
0 "mdl":    WAVE
1 "att":    linf [-15, 15, 61] dB, attack
2 "sust":   linf [-24, 24, 97] dB, sustain
3 "g":      linf [-18, 9, 55] dB, gain
```



Auto Rider Dynamics

```
0 "mdl":    RIDE
1 "thr":    linf [-54, 18, 73] dB, thr
2 "tgt":    linf [-48, 0, 97] dB, target
3 "spd":    int [1…50] speed
4 "ratio":  flt [2.0, 4.0, 8.0,
                 20.0, 100.0] ratio
5 "hld":    logf [.1, 10, 65] s, hold
6 "range":  linf [1, 15, 29] dB, range
```

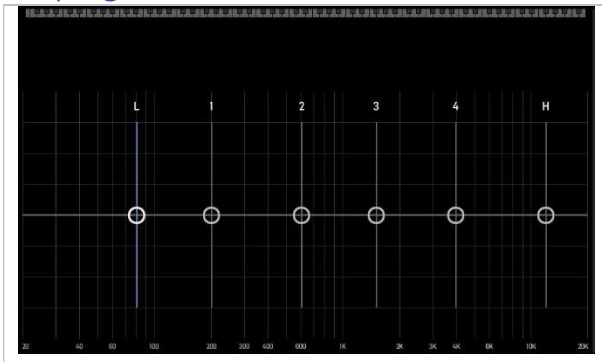
Soul Warmth Preamp

```
0 "mdl":    WARM
1 "drv":    linf [10, 100, 91] %, drive
2 "hrm":    linf [-100, 100, 201] harm
3 "col":    linf [-1, 1, 41] color
3 "trim":   linf [-18, 6, 49] dB, trim
4 "mix":    linf [0, 100, 101] dB, mix
```

Dynamic EQ

```
0 "mdl":    DEQ
1 "thr":    linf [-60, 0, 121] dB, thr
2 "ratio":  flt [1.2, 1.3, 1.5, 2.0,
                 3.0, 5.0, 10.0] ratio
3 "att":    linf [0, 200, 201] ms, attack
4 "rel":    logf [20, 4000, 130] ms, release
5 "filt":   str [OFF, BP, LP6, LP12,
                 HP6, HP12] filter
6 "g":      linf [-15, 15, 301] dB, gain
7 "f":      logf [20, 20000, 961] Hz, freq
8 "q":      logf [.442, 10, 181] q
9 "mode":   str [low, high] mode
```

EQ plugins



Standard EQ

Channel:

```
 0 "mdl":  STD
 1 "lg":   linf [-15, 15, 301] dB, gain l
 2 "lf":   logf [20, 2000, 641] Hz, freq l
 3 "lq":   logf [0.442, 10, 181] q l
 4 "leq":  str [SHV, PEQ] eq l
 5 "1g":   linf [-15, 15, 301] dB, gain 1
 6 "1f":   logf [20, 20000, 961] Hz, freq 1
 7 "1q":   logf [0.442, 10, 181] q 1
 8 "2g":   linf [-15, 15, 301] dB, gain 2
 9 "2f":   logf [20, 20000, 961] Hz, freq 2
10 "2q":   logf [0.442, 10, 181] q 2
11 "3g":   linf [-15, 15, 301] dB, gain 3
12 "3f":   logf [20, 20000, 961] Hz, freq 3
13 "3q":   logf [0.442, 10, 181] q 3
14 "4g":   linf [-15, 15, 301] dB, gain 4
15 "4f":   logf [20, 20000, 961] Hz, freq 4
16 "4q":   logf [0.442, 10, 181] q 4
17 "hg":   linf [-15, 15, 301] dB, gain h
18 "hf":   logf [50, 20000, 833] Hz, freq h
19 "hq":   logf [0.442, 10, 181] q h
20 "heq":  str [SHV, PEQ] eq h
```

Bus, mtx, main:

```
 0 "mdl":  STD
 1 "lg":   linf [-15, 15, 301] dB, gain l
 2 "lf":   logf [20, 2000, 641] Hz, freq l
 3 "lq":   logf [0.442, 10, 181] q l
 4 "leq":  str [SHV, PEQ, CUT] eq l
 5 "1g":   linf [-15, 15, 301] dB, gain 1
 6 "1f":   logf [20, 20000, 961] Hz, freq 1
 7 "1q":   logf [0.442, 10, 181] q 1
 8 "2g":   linf [-15, 15, 301] dB, gain 2
 9 "2f":   logf [20, 20000, 961] Hz, freq 2
10 "2q":   logf [0.442, 10, 181] q 2
11 "3g":   linf [-15, 15, 301] dB, gain 3
12 "3f":   logf [20, 20000, 961] Hz, freq 3
13 "3q":   logf [0.442, 10, 181] q 3
14 "4g":   linf [-15, 15, 301] dB, gain 4
15 "4f":   logf [20, 20000, 961] Hz, freq 4
16 "4q":   logf [0.442, 10, 181] q 4
17 "5g":   linf [-15, 15, 301] dB, gain 5
18 "5f":   logf [20, 20000, 961] Hz, freq 5
19 "5q":   logf [0.442, 10, 181] q 5
20 "6g":   linf [-15, 15, 301] dB, gain 6
21 "6f":   logf [20, 20000, 961] Hz, freq 6
22 "7q":   logf [0.442, 10, 181] q 6
23 "hg":   linf [-15, 15, 301] dB, gain h
24 "hf":   logf [50, 20000, 833] Hz, freq h
25 "hq":   logf [0.442, 10, 181] q h
26 "heq":  str [SHV, PEQ, CUT] eq h
27 "tilt": linf [-6, 6, 49] dB, tilt
```

Soul Analog EQ

```
 0 "mdl":   SOUL
 1 "mix":   linf [0, 125, 126] %, mix
 2 "lf":    linf [0, 10, 101] lo freq
 3 "lg":    linf [-5, 5, 101] lo gain
 4 "lmf":   linf [0, 10, 101] lm freq
 5 "lmf3":  int [0, 1] lm /3
 6 "lmq":   linf [0, 10, 101] lm q
 7 "lmg":   linf [-5, 5, 101] lm gain
 8 "hmf":   linf [0, 10, 101] hm freq
 9 "hmf3":  int [0, 1] hm x3
10 "hmq":   linf [0, 10, 101] hm q
11 "hmg":   linf [-5, 5, 101] hm gain
12 "hf":    linf [0, 10, 101] hf freq
13 "hg":    linf [-5, 5, 101] hf gain
```

Even 88-Formant EQ

```
 0 "mdl":   E88
 1 "mix":   linf [0, 125, 126] %, mix
 2 "lf":    linf [0, 10, 101] lf freq
 3 "lg":    linf [-5, 5, 101] lf gain
 4 "lq":    str [LOW, HIGH] lf q
 5 "lt":    str [BELL, SHELV] lf type
 6 "lmf":   linf [0, 10, 101] lm freq
 7 "lmg":   linf [-5, 5, 101] lm gain
 8 "lmq":   linf [0, 10, 101] lm q
 9 "hmf":   linf [0, 10, 101] hm freq
10 "hmg":   linf [-5, 5, 101] hm gain
11 "hmq":   linf [0, 10, 101] hm q
12 "hf":    linf [0, 10, 101] hm freq
13 "hg":    linf [-5, 5, 101] hf gain
14 "hq":    str [LOW, HIG] hf q
15 "ht":    str [BELL, SHELV] hf type
```

Even 84 EQ

```
 0 "mdl":   E84
 1 "mix":   linf [0, 125, 126] %, mix
 2 "g":     linf [-20, 20, 81] dB, gain
 3 "lf":    str [OFF, 35, 60, 110, 220] lf freq
 4 "lg":    linf [-5, 5, 101] lf gain
 5 "mf":    str [OFF, 350, 700, 1k6, 3k2,
                 4k8, 7k2] mid freq
 6 "mg":    linf [-5, 5, 101] mid gain
 7 "mq":    str [LOW, HIGH] mid q
 8 "hf":    str [10k, 12k, 16k, OFF] hf freq
 9 "hg":    linf [-5, 5, 101] hf gain
```

Focusrite ISA 110 EQ

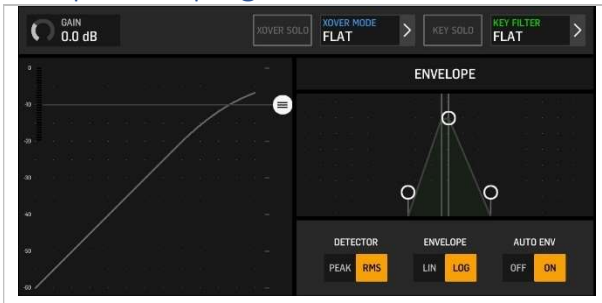
```
 0 "mdl":   F110
 1 "mix":   linf [0, 125, 126] %, mix
 2 "peq":   int [0, 1] peq on
 3 "lmf":   linf [0, 10, 101] lm freq
 4 "lmg":   linf [-5, 5, 101] lm gain
 5 "lmq":   linf [0, 10, 101] lm q
 6 "lmf3":  int [0, 1] lm /3
 7 "hmf":   linf [0, 10, 101] hm freq
 8 "hmg":   linf [-5, 5, 101] hm gain
 9 "hmq":   linf [0, 10, 101] hm q
10 "hmf3":  int [0, 1] hm x3
11 "shv":   inf [0, 1] shv on
12 "lf":    str [33, 56, 95, 160,
                270, 460] lf freq
13 "lg":    linf [-5, 5, 101] lf gain
14 "hf":    str [3k3, 4k7, 6k8, 10k,
                 15k, 18k] hf freq
15 "hg":    linf [-5, 5, 101] hf q
16 "g":     linf [-18, 18, 73] gain
```

Pulsar P1a/M5 EQ

```
 0 "mdl":   PULSAR
 1 "mix":   linf [0, 125, 126] %, mix
 2 "eq1":   int [0, 1] eq1 on
 3 "1lb":   linf [0, 10, 101] lf boost
 4 "1latt": linf [0, 10, 101] lf att
 5 "1lf":   str [20, 30, 60, 100] Hz, lf freq
 6 "1hw":   linf [0, 10, 101] hf wid
 7 "1hb":   linf [0, 10, 101] hf boost
 8 "1hf":   str [3k, 4k, , 5k, 8k, 10k,
                 12k, 16k] Hz, hf freq
 9 "1hatt": linf [0, 10, 101] hf att
10 "1hattf":str [5k, 10k, 20k] hf att
11 "eq5":   int [0, 1] eq5 on
12 "5lb":   linf [0, 10, 101] lm boost
13 "5lf":   str [200, 300, 500, 700,
```

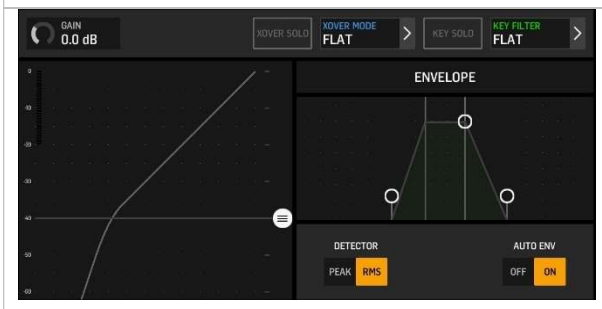
	1k] Hz, lf freq 14 **"5md"**: linf [0, 10, 101] mid dip 15 **"5mf"**: str [200, 300, 500, 700, 1k, 1k5, 2k, 3k, 4k, 5k, 7k] Hz, mid freq 16 **"5hb"**: linf [0, 10, 101] HM boost 17 **"5hf"**: str [1k5, 2k, 3k, 4k, 5k] Hz, hf freq
	Mach EQ4 0 **"mdl"**: MACH4 1 **"mix"**: linf [0, 125, 126] %, mix 2 **"sub"**: linf [-5, 5, 101] sub 3 **"40"**: linf [-5, 5, 101] 40 4 **"160"**: linf [-5, 5, 101] 160 5 **"650"**: linf [-5, 5, 101] 650 6 **"2k5"**: linf [-5, 5, 101] 2k5 7 **"air"**: linf [0, 10, 101] air 8 **"airm"**: str [OFF, 2k5, 5k, 10k, 20k, 40k] air mode 9 **"again"**: int [0, 1] auto

Compressor plugins



Standard compressor

```
 0 “mdl”:   COMP
 1 “mix”:   linf [0, 100, 101] %, mix
 2 “gain”:  linf [-6, 12, 37] dB, gain
 3 “thr”:   linf [-60, 0, 121] dB, thr
 4 “ratio”: flt [1.1, 1.2, 1.3, 1.5, 1.7, 2.0,
                 2.5, 3.0, 3.5, 4.0, 5.0, 6.0,
                 8.0, 10., 20., 50., 100.] ratio
 5 “knee”:  int [0…5] knee
 6 “det”:   str [PEAK, RMS] detector
 7 “att”:   linf [0, 120, 121] ms, attack
 8 “hld”:   linf [1, 200, 200] ms, hold
 9 “rel”:   logf [4, 4000, 130] ms release
10 “env”:   str [LIN, LOG] envelope
11 “auto”:  int [0, 1] auto
```



Standard expander

```
 0 “mdl”:   EXP
 1 “mix”:   linf [0, 100, 101] %, mix
 2 “gain”:  linf [-6, 12, 37] dB, gain
 3 “thr”:   linf [-60, 0, 121] dB, thr
 4 “ratio”: flt [1.1, 1.2, 1.3, 1.5, 1.7, 2.0,
                 2.5, 3.0, 3.5, 4.0, 5.0, 6.0,
                 8.0, 10., 20., 50., 100.] ratio
 5 “knee”:  int [0…5] knee
 6 “det”:   str [PEAK, RMS] detector
 7 “att”:   linf [0, 120, 121] ms, attack
 8 “hld”:   linf [1, 200, 200] ms, hold
 9 “rel”:   logf [4, 4000, 130] ms release
10 “env”:   str [LIN, LOG] envelope
11 “auto”:  int [0, 1] auto
```

BDX 160 Compressor/Limiter

```
 0 “mdl”:   B160
 1 “mix”:   linf [0, 100, 101] %, mix
 2 “gain”:  linf [-6, 12, 37] dB, gain
 3 “thr”:   logf [.01, 5, 65] thr
 4 “ratio”: flt [1.1, 1.2, 1.3, 1.5, 1.7, 2.0,
                 2.5, 3.0, 3.5, 4.0, 5.0, 6.0,
                 8.0, 10., 20., 50.] ratio
```

BDX 560 Easy Compressor

```
 0 “mdl”:   B560
 1 “mix”:   linf [0, 100, 101] %, mix
 2 “gain”:  linf [-6, 12, 37] dB, gain
 3 “thr”:   linf [-40, 20, 121] dB, thr
 4 “ratio”: flt [1.1, 1.2, 1.5, 2.0, 3.0, 4.0,
                 5.0, 7.0, 10., 50., 999.,
                 -5.0, -3.0, -2.0, -1.0] ratio
 5 “auto”:  int [0, 1] auto
```

Draw More Compressor

```
 0 “mdl”:   D241
 1 “mix”:   linf [0, 100, 101] %, mix
 2 “gain”:  linf [-6, 12, 37] dB, gain
 3 “thr”:   linf [0, -60, 121] dB, thr
 4 “ratio”: flt [1.1, 1.2, 1.3, 1.5,
                 1.7, 2.0, 3.0, 3.5,
                 4.0, 5.0, 6.0, 8.0,
                 10.0, 20.0, 50.0,
                 100.0] ratio
 5 “att”:   linf [.5, 100, 65] ms, attack
 6 “rel”:   logf [50, 5000, 130] ms release
 7 “lim”:   linf [-20, 0, 41] dB, lim thr
 8 “lrel”:  logf [50, 5000, 130] ms, lim rel
 9 “auto”:  int [0, 1] auto
```

Red Compressor

```
 0 "mdl":    RED3
 1 "mix":    linf [0, 100, 101] %, mix
 2 "gain":   linf [-6, 12, 37] dB, gain
 3 "thr":    linf [-48, 0, 97] dB, thr
 4 "ratio":  flt [1.1, 1.2, 1.3, 1.5, 2.0,
                  2.5, 3.0, 3.5, 4.0, 5.0,
                  6.0, 8.0, 10.] ratio
 5 "att":    linf [1, 50, 65] ms, attack
 7 "rel":    logf [100, 4000, 65] ms release
 8 "auto":   int [0, 1] auto
```

Soul 9000 Channel Compressor

```
 0 "mdl":    9000C
 1 "mix":    linf [0, 100, 101] %, mix
 2 "gain":   linf [-6, 12, 37] dB, gain
 3 "thr":    linf [-48, 0, 97] dB, thr
 4 "ratio":  flt [1.3, 1.43, 1.57, 1.8, 2.0,
                  2.8, 3.3, 4.0, 5.0 , 6.0,
                  7.0, 9.0, 12.0, 20.0, 50.0,
                  100.0] ratio
 5 "fast":   int [0, 1] fast att
 6 "rel":    logf [100, 4000, 65] ms release
 7 "peak":   int [0, 1] peak
```

Soul G Buss Compressor

```
 0 "mdl":    SBUS
 1 "mix":    linf [0, 100, 101] %, mix
 2 "gain":   linf [-6, 12, 37] dB, gain
 3 "thr":    linf [-48, 0, 81] dB, thr
 4 "ratio":  flt [1.5, 2.0, 3.0, 4.0, 5.0,
                  10.0] ratio
 5 "att":    flt [0.1, 0.3, 1.0, 3.0, 10.0,
                  30.0] ratio
 6 "rel":    str [0.1, 0.2, 0.4, 0.8, 1.6,
                  AUTO] release
```

Even Compressor/Limiter

```
 0 "mdl":    ECL33
 1 "mix":    linf [0, 100, 101] %, mix
 2 "gain":   linf [-6, 12, 37] dB, gain
 3 "lon":    int [0, 1] lim on
 4 "lthr":   linf [-12, 0, 25] dB, lim thr
 5 "lrec":   str [50, 100, 200, 800,
                  A1, A2] lim rec
 6 "lfast":  int [0, 1] lim fast
 7 "con":    int [0, 1] comp on
 8 "cthr":   linf [-35, -5, 61] dB, comp thr
 9 "ratio":  str [1.5, 2.0. 3.0, 4.0, 6.0] ratio
10 "crec":   str [100, 400, 800, 1500
                  A1, A2] comp rec
11 "cfast":  int [0, 1] comp fast
```

Eternal Bliss

```
 0 "mdl":    BLISS
 1 "mix":    linf [0, 100, 101] %, mix
 2 "gain":   linf [-6, 12, 37] dB, gain
 3 "thr":    linf [-50, 0, 101] dB, thr
 4 "ratio":  flt [1.2, 1.3, 1.6, 2.0, 3.0,
                  -1.0, -2.0, -3.0,  -4.0] ratio
 5 "att":    linf [.4, 150, 65] ms, attack
 6 "rel":    logf [5, 1200, 65] ms release
 7 "afast":  int [0, 1] auto fast
 8 "alog":   int [0, 1] anti log
 9 "glon":   int [0, 1] gr limit on
10 "glim":   linf [-21, 0, 43] gr limit
```

Amplifier76 Limiting Amplifier

```
0 "mdl":    76LA
1 "mix":    linf [0, 100, 101] %, mix
2 "gain":   linf [-6, 12, 37] dB, gain
3 "in":     linf [-48, 0, 97] dB, input
4 "out":    linf [-48, 0, 97] dB
5 "att":    linf [1, 7, 61] attack
6 "rel":    linf [1, 7, 61] release
7 "ratio":  str [4, 8, 12, 20, ALL] ratio
```

Leveling Amplifier 2A

```
0 "mdl":    LA
1 "mix":    linf [0, 100, 101] %, mix
2 "gain":   linf [-6, 12, 37] dB, gain
3 "ingain":linf [0, 100, 101] gain
4 "peak":   linf [0, 100, 101] peak
5 "mode":   str [comp, lim] mode
```

Fairkid Model 670

```
0 "mdl":    F670
1 "mix":    linf [0, 100, 101] %, mix
2 "gain":   linf [-6, 12, 37] dB, gain
3 "in":     linf [-20, 0, 81] dB, input
4 "thr":    linf [0, 10, 41] thr
5 "time":   int [1…6] time
6 "bias":   linf [0, 1, 101] bias
```

No Stressor

```
0 "mdl":    NSTR
1 "mix":    linf [0, 100, 101] %, mix
2 "gain":   linf [-6, 12, 37] dB, gain
3 "in":     linf [0, 10, 101] input
4 "ou":     linf [0, 10, 101] output
5 "att":    linf [0, 10, 101] attack
6 "rel":    linf [0, 10, 101] release
7 "ratio":  str [1.5:1, 2:1, 3:1, 4:1, 6:1,
                 10:1, 20:1, NUKE] ratio
```

PIA 2250

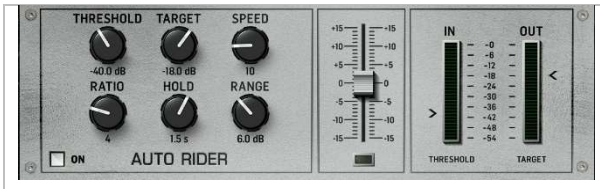
```
0 "mdl":    2250
1 "mix":    linf [0, 100, 101] %, mix
2 "gain":   linf [-6, 12, 37] dB, gain
3 "thr":    linf [0, 10, 101] threashold
4 "ratio":  linf [0, 10, 101] output
5 "att":    str [FAST, MED, SLOW] attack
6 "rel":    logf [50, 3000, 130] ms, release
7 "knee":   str [HARD, SOFT] knee
8 "Type":   str [OLD, NEW] type
```

LTA100 Leveler

```
0 "mdl":    L100
1 "mix":    linf [0, 100, 101] %, mix
2 "gain":   linf [-6, 12, 37] dB, gain
3 "ingain":linf [0, 10, 101] gain
4 "gr":     linf [0, 10, 101] gain reduction
5 "att":    str [FAST, MED, SLOW] attack
6 "rel":    str [FAST, MED, SLOW] release
```

Wave Designer

```
0 "mdl":    WAVE
1 "mix":    linf [0, 100, 101] %, mix
2 "gain":   linf [-6, 12, 37] dB, gain
3 "att":    linf [-15, 15, 61] dB, attack
4 "sust":   linf [-24, 24, 97] dB, sustain
5 "g":      linf [-16, 9, 55] dB, gain
```



Auto Rider Dynamics

```
0 "mdl":   RIDE
1 "mix":   linf [0, 100, 101] %, mix
2 "gain":  linf [-6, 12, 37] dB, gain
3 "thr":   linf [-54, 18, 73] dB, thr
4 "tgt":   linf [-48, 0, 97] dB, target
5 "spd":   int [1…50] speed
6 "ratio": flt [2.0, 4.0, 8.0,
                20.0, 100.0] ratio
7 "hld":   logf [.1, 10, 65] s, hold
8 "range": linf [1, 15, 29] dB, range
```

Appendix: Routing

Routing is a key aspect in digital consoles and can be … intimidating, especially with desks such as WING, with a multitude of physical sources and destination, tap points, and total flexibility of signal path arrangements for mixing and routing. "*In the world of digital consoles with a multitude of inputs, outputs, and complex mixing capabilities, routing is the fundamental process that determines how audio signals flow throughout the system. It's akin to a sophisticated traffic control center, ensuring each sound reaches its intended destination with the desired processing applied. Routing in digital consoles is the foundation for achieving a high-quality, well-controlled sound. By mastering this skill, you can unlock the full potential of your console and create professional-sounding mixes with unparalleled flexibility*"[55].

The following chapters along with existing videos on routing you can find on the web will help you with your first steps in routing your signals in the WING console.

WING routing is always done from a WING perspective:

- For input routing, input `SOURCES` are the physical connections to WING, while destinations are either `WING CHANNELS` or `OUTPUTS` (i.e. physical outputs);
- For output routing, signal `SOURCES` are any of the possible tap points in WING to send out a digital audio signal, including `INPUT SOURCES`, `BUS`, `MAIN`, `MATRIX`, `USER SIGNAL`, `MONITOR`, and `FX SENDS`, while destinations are the physical outputs available from the desk or additional devices that are connected to it.

Benefits of Effective Routing:

- **Clean Mixes**: Proper routing avoids unwanted signal bleed and ensures each element sits clearly within the mix.
- **Efficient Workflow**: By creating custom routing setups, you can save time during live performances or studio sessions.
- **Creative Possibilities**: Advanced routing unlocks creative options like sending specific instruments to dedicated effects mixes, or managing both FOH and Monitoring from the same desk.

Understanding Routing Interfaces:
Digital consoles such as WING offer visual interfaces for routing, with screens depicting virtual "patches" connecting inputs, outputs, and internal processing modules. For WING, the interfaces are the main touchscreen using the `ROUTING` dedicated button, or software applications such as `WING-Edit`[56] or `Mixing Station`[57].

[55] Source: Gemini AI

[56] See: https://www.behringer.com/series.html?category=R-BEHRINGER-WINGSERIES, under the Software section

[57] See: https://mixingstation.app/

Input Routing

WING has numerous possibilities when it comes to connecting sources and channels (so called “routing”), effectively enabling audio to ‘flow’ from its source to the WING audio engine for processing and mixing within the desk.

There is a very large choice of no less than 376 input sources that can be found under the ROUTING→SOURCES screen, the SOURCE GROUP selection includes:

- LOCAL IN (8 local XLR inputs on the full-size desk)
- AUX IN (8 local 6.3mm inputs on the full-size desk)
- AES/EBU IN (2 AES/EBU inputs)
- OSCILLATOR (2 internal oscillator sources with various signal options and settings)
- AES50-A, B and C (each with 48 inputs)
- ST CONNECT (StageConnect™, configurable 32 IN or OUT at line level on a standard XLR/DMX cable)
- USB AUDIO (48 inputs from a USB-2.0 port)
- WLIVE PLAY (2x 32 inputs from one or two SD cards)
- DANTE[58] (64 inputs from either a card or internal module, or 128 inputs if both are installed)
- USB PLAYER (4 inputs from the USB stick input)
- USER SIGNAL (48 configurable user data path or patches overlapping/referencing sources above)

All Sources above come with their associated SETTINGS *(Mono/Stereo/MS, Gain, Polarity, Mute, Phantom, …)*, ICON, COLOR, and TAGS.

The process of “source routing” or “input routing” is the action of associating a SOURCE to one of the WING 48 (input/aux) Channels (or Channel Strips) for mixing their audio within the desk. This is accomplished by pressing the ROUTING button on the left of the WING Screen, and selecting the CHANNELS tab on the screen, opening the following screen:

There are two routing options: MAIN and ALT. Both serve the same purpose and can be selected within Channel Strips; There are therefore two routing tables available at the desk.
WING routing tables are write-protected (to avoid major issues during live performances) unless the unlocked

[58] This could be another option

padlock [🔓] is selected. With the padlock being green, a Channel Strip can be selected on the left side of the screen and a SOURCE entry can be selected from the Sources available on the right side of the screen.

Different groups of SOURCES blocks are available from the SOURCE GROUP pull down menu (see below):

SOURCES include actual HW sources and logical audio paths such as BUS, MAIN, MATRIX and USER SIGNAL, which are either the result of partially mixed audio or a specific/customized selection of Source in the case of USER SIGNAL.

After a console init as shown above, LOCAL IN 1…8 mono sources will already be routed to Channels 1…8, USB AUDIO 1&2 will be combined as a stereo source routed to Aux 1, and Aux 2 will receive USB PLAYER 1&2 as a stereo source. This constitutes the default MAIN routing table. The ALT routing table is empty.

For example, routing WLIVE PLAY sources 1..16 to Channels 9..24 can be done by a click on the +1AUTO button, selecting the first Channel to modify routing for (9), selecting WLIVE PLAY in the SOURCE GROUP pull-down menu and sequentially clicking on the 16 first entries of WLIVE PLAY, resulting in the following screen (and MAIN routing table).

Additional, similar, or different choices for routing could be done for the ALT routing table, offering an alternate set of SOURCES to mix from at the mixing desk. Note that the MAIN/ALT selection at the Channel Strip level is accomplished by selecting either the MAIN or the ALT source for that Channel, i.e. moving from one to the other will possibly select a different source to mix, losing the previous one.

At that point, Channel Strips 9 to 24 can be used to mix the audio data issuing from the first 16 tracks of SD card 1[59]. Channel Strip 1 to 8 can be used to mix the audio data coming from local inputs 1 to 8 at the back of the console, and Aux strips 1 and 2 can be used to mix the audio signals from USB 1 & 2 and the 2 tracks from the USB stick player.

What if the 16 **WLIVE PLAY** tracks above were representing 8 distinct stereo channels?

WLIVE PLAY sources must be declared as stereo pairs; This is done by returning to the **SOURCES** tab and selecting the **WLIVE PLAY** source group, showing all 64 possible entries. Selecting entry **1** and clicking on **STEREO** in the **SETTINGS** will automatically 'join' entries **1** and **2** as a stereo pair named **CRD1/L** and **CRD2/R**, the same action can then be done for entries **3, 5, 7, 9, 11, 13**, and **15**, resulting in 8 pairs of stereo sources that can be routed to 8 different channels as described earlier. The pictures below show the screens resulting from the actions we just described.

As a result from the operations above, Channel Strips 9 to 16 can now be used to mix the audio data issued from the first 16 tracks of SD card 1. Channel Strip 1 to 8 can still be used to mix the audio data coming from local inputs 1 to 8 at the back of the console, and Aux strips 1 and 2 can still be used to mix the audio signals from USB 1 & 2 and the 2 tracks from the USB stick player.

[59] Note that SD card 1 maps to entries 1..32, and SD card 2 maps to entries 33..64 in the **WLIVE PLAY** or **REC** screens

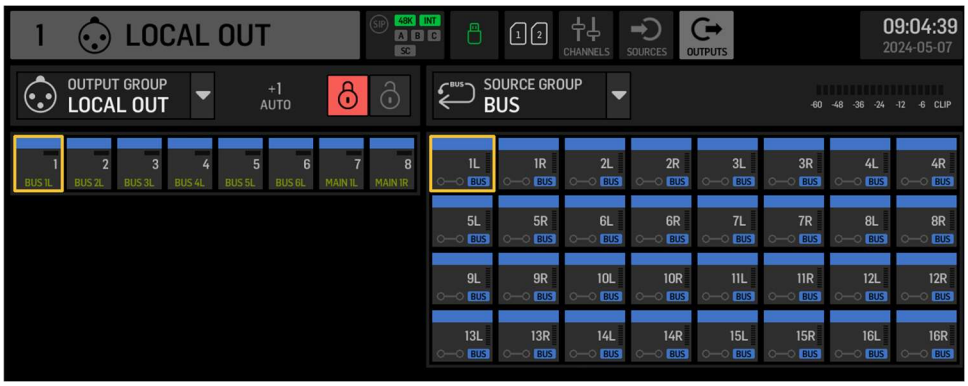
Output Routing

Output routing works in a similar way to Input routing. This time though, the OUTPUTS tab is selected in the ROUTING screen, revealing OUTPUT GROUPS from a pull-down menu, and representing the physical outputs where audio signals from the console can be routed to, using digital audio sources selected from the SOURCE GROUPS pull-down menu and entries.

The selection of output physical connections is as numerous as for inputs and characterizes the console versatility and extended capabilities with 374 physical outputs that can be found under the ROUTING→OUTPUT screen; The OUTPUT GROUP selection includes:

- LOCAL OUT (8 local XLR outputs on the full-size desk)
- AUX OUT (8 local 6.3mm outputs on the full-size desk)
- AES/EBU OUT (2 AES/EBU outputs)
- AES50-A, B and C (each with 48 outputs)
- ST CONNECT (StageConnect™, configurable 32 IN or OUT at line level on a standard XLR/DMX cable)
- USB AUDIO (48 outputs from a USB-2.0 port)
- WLIVE REC (2x 32 outputs to one or two SD cards)
- DANTE[60] (64 outputs from either a card or internal module, or 128 outputs if both are installed)
- RECORDER (4 outputs to the USB stick input)

As for source assignments, the console comes with a default output routing right after being initialized. This is shown below with LOCAL OUT 1..8 receiving BUS 1L to BUS 6L, MAIN 1L and MAIN 1R, respectively. Note also that AUX OUT 7&8 are receiving MONITOR A&B by default.



Changing the output routing is made from the ROUTING→OUTPUTS screen, with first clicking on the OUTPUT GROUP of interest to select the first physical destination you want to assign a WING signal to.

Say we would like to record the first 8 (all stereo) channels of our show as a new session onto SD card 2, along with the resulting show mix on the USB stick of the front panel as a 2-track mix coming from MAIN1; The operations one would perform are as follows:

After selecting the WLIVE REC group of physical output on the left side of the screen and clicking on any entry in that panel, we need to select where routed signals will be coming from. That selection is possible from the SOURCE GROUP pull-down menu on the right side of the screen that offers a list of all possible 'tap points' for getting digital signals from the desk to physical outputs. In our example, we would select the audio signal sources that feed our Channel Strips 1 to 8. Let's further assume our 8 Channel Strips are taking their inputs from WLIVE PLAY as described above. SD card 2 maps its 32 entries from 33 to 64 in the WLIVE PLAY or REC

[60] Could be another option

panels. As we did for input routing, we first select the unlocked padlock [🔓], click on the first entry for SD card 2 (`CRD 33`) in the `WLIVE REC` panel on the left side of the screen, click on the `+1AUTO` button for ease of selection and choose our audio signals to route to our outputs by clicking on the 16 entries in the `WLIVE PLAY` panel on the right side of the screen, starting at entry `CRD1`, resulting in the following routing table:

The audio data used for our mix will be recorded to SD card 2, from SD card 1 (only when engaging record on SD card 2 and play on SD card 1, of course).

We also need to set the output routing for recording our live mix; In the `ROUTING→OUTPUTS` screen, with clicking on the `OUTPUT GROUP` on the left side of the screen, we select `RECORDER` and click on the first entry, `1`. With the unlocked padlock [🔓] selected and the `+1AUTO` button engaged, we select `MAIN` in the `SOURCE GROUP` on the right side of the screen, and then click on the `1L` and `1R` entries in the displayed table. This completes our output routing with the following screen:

We can now select the USB recorder and start a 2-channel recording on USB stick, then select the SD card screen, start recording on card 2, select our 8 stereo tracks session and hit play on card 1 and mix our session, simultaneously getting a digital copy of our dry data from SD1 to SD2 and a live mix result as a stereo wav file in the USB stick.

Advanced Routing Options

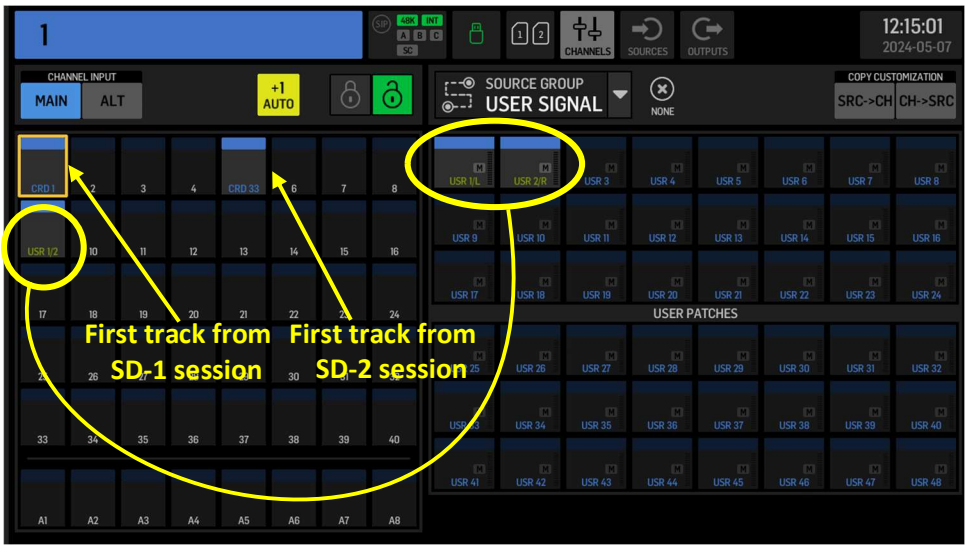
All routing scenarios presented above have a restriction when it comes to stereo pairs. The HW limitations of the desk impose stereo pairs to always be in the form [odd-even] SOURCE numbers; i.e. you cannot route a stereo signal to a single channel strip if your two mono sources are connected to say LOCAL IN 2 and LOCAL IN 3, or if they are connected to WLIVE PLAY 1 and WLIVE PLAY 33.

This is where USER SIGNALS come into play , leveraging the internal FPGA routing chip flexibility to remove some of that restriction. A USER SIGNAL is a virtual channel, and proposes two variants: USER SIGNAL and USER PATCHES which we'll detail below:

USER SIGNAL

A USER SIGNAL can only accept INPUT, AUX or BUS, MAIN or MATRIX channels as source. Setting or assigning sources to USER SIGNAL is done with selecting the ROUTING->SOURCES screen and choosing one of the 24 USER SIGNAL entries in the SOURCE GROUP pull-down menu on the left side of the screen.

Clicking on an entry will display the SOURCE characteristics on the right side of the screen, with SOURCE SETTINGS, ICON, COLOR, POLARITY, and MUTE, and a +ASSIGN button that is used for selecting which channel, tap point (TAP or POST) and whether using stereo, mono, or M/S data as signal(s) for the selected USER SIGNAL entry. If the selected USER SIGNAL entry is stereo, it is possible to choose two totally disjoint sources for each of the L and R paths of the selected USER SIGNAL, such as for example channel 1 and channel 5 that would be routed with WLIVE PLAY 1 and WLIVE PLAY 33 to take our example above, creating a stereo pair that can now be assigned/routed to a single Channel strip thanks to USER SIGNALS. The screenshots below show routing displays for such a case , with channels strips 1 and 5 routed to sources WLIVE PLAY 1 & 5, and channel strip 9 routed to stereo USER SIGNAL 1 that routes channels 1 and 5 as a single stereo pair to itself.



That same `USER SIGNAL 1` can also be used as a `SOURCE` for an `OUTPUT`, such as for example to record into a DAW on a PC using a USB connection, as a single stereo pair into `USB 1&2`.

To achieve this, we would click on the first entry of `USB AUDIO` in the `OUTPUT GROUP` on the left side of the `ROUTING→OUTPUTS` screen. With the unlocked padlock [🔓] selected and the `+1AUTO` button engaged, we select `USER SIGNAL` as a `SOURCE GROUP` on the right side of the screen and click on the `USER 1L` and `USER 2R` entries in the displayed table. This completes our output routing with the following screen:

We can then hit play on WLIVE sessions on both SD card 1 and 2, and will get our signals available as a single stereo pair for recording a 2-track session over USB cable to a connected PC.

USER PATCH

A USER PATCH can be used the same way as a USER SIGNAL but unlike USER SIGNAL, the audio physical SOURCE will directly connect to a USER PATCH, thus bypassing the need for using an intermediate channel strip or set of channel strips.

Setting or assigning sources to USER PATCH is done with selecting the ROUTING->SOURCES screen and choosing USER SIGNAL in the SOURCE GROUP pull-down menu on the left side of the screen and selecting one of the 24 USER PATCH entries in the list. As for USER SIGNAL, clicking on a USER PATCH entry will display the SOURCE characteristics on the right side of the screen, with SOURCE SETTINGS, ICON, COLOR, POLARITY, and MUTE, and a +ASSIGN button that is used for selecting which physical SOURCE will be used. A USER PATCH can accept any of LOCAL IN, AUX IN, AES/EBU IN, AES50 A/B/C, ST CONNECT, USB AUDIO, WLIVE PLAY, DANTE, or USB PLAYER signal as its routed SOURCE.

If the selected USER PATCH entry is stereo, it is possible to choose two totally disjoint sources for each of the L and R paths of the selected USER PATCH, such as routing for example WLIVE PLAY 1 and WLIVE PLAY 33 to USER PATCH stereo entry 26L/26R to use once more our example above, creating a stereo pair that can now be assigned/routed to a single Channel strip.

The screenshots below show routing displays for such a case, with sources WLIVE PLAY 1 & 33 routed to USER SIGNAL 25L/26R, itself routed as a single stereo pair into channel 9.

USER PATCH has the advantage of routing simplicity over USER SIGNAL. On the other hand, USER SIGNAL offers more signal processing or mixing capabilities over USER PATCH, to the expense of using intermediate Channels.

Appendix: Shows, Scenes (Snaps, Snippets, Presets & Audio Clips)

The WING desk has a high level of functionality to manage record and restore shows, snaps, snippets and presets, or scenes.

Shows

A key feature in digital consoles is the ability to save and restore state (in different forms) to easily change from one set to another, or save work for later use. This helps maximize the use of the desk in situations where several bands share the same console, or in recording studios where saving console state is a must have for effectively managing recordings and customer data.

For WING, a **Show** is typically a collection of **Scenes**. Show files contain references to Scene entities, and not the actual data.

Shows can be managed directly from the WING screen via the LIBRARY button. One can create a Show, open, or delete it. There can be only a single active Show at any time.
Snaps, Snippets, FX or Channel Presets or Audio Clips can be added to the current Show, they can also be re-organized using the options provided in the LIBRARY section. When added to a Show file, they are referenced as Scenes.

Users can 'navigate' up and down [i.e. loading Scenes] in the current Show using the Show Control buttons dedicated to that effect [GO, NEXT, PREV]. Some of the Show items can also be marked as 'skip' for a quick avoiding loading them during navigation. Show items can also be marked with a 'link' tag to enable simultaneous loading of multiple items during navigation.

Please refer to the Behringer documents on how to use Shows[61]

Scenes

A Scene can represent anything used in a Show. It can refer to a Snap, a Snippet, a Channel of FX Preset or an Audio Clip, or a combination thereof. Each single entry in a Show file is a separate Scene that can be loaded using the Library navigation options.

Snaps

A Snap file contains the full set of WING parameters, optionally associated with a Scope that lists the set of parameters of interest at the time the Snap was created. Scopes can be changed at load time if needed. They can also be modified as needed and saved.

[61] Available at: https://mediadl.musictribe.com/download/software/behringer/WING/WING_Firmware_1.13_GUI-Description.pdf

Snippets

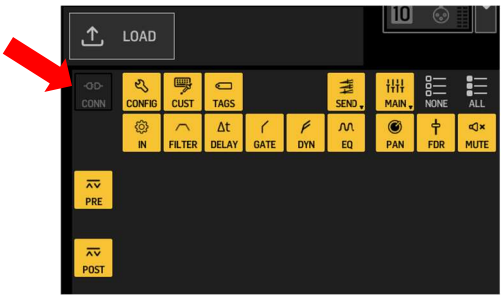
A Snippet file allows recording of any WING parameter changes as well as manually adding/removing of parameters using the Library buttons ADD ITEMS and REMOVE ITEMS.

- REC FOCUS defines which parameters are observed during REC active.
- LOAD FOCUS allows loading a parameter set from any existing snippet.
- When a snippet is saved or updated, the *current* values of all parameters (in FOCUS) are written to the file and cannot be changed once recorded to INT or USB file.

Presets

- FX and CHANNEL Presets
    - Presets contain target FX slot(s) and target channel/scope information; When used within a Show, the Preset data is instantiated within the Show as a Scene, and the same Preset can be used to load different FX engines / channels (with different scope)). After adding a Preset to a Show, just change settings and click UPDATE SCENE (don't forget to save the show).
    - If you set the target FX slot of a Channel Preset (one of the inserts) to NONE, the insert is switched off when loading the Preset.
    - Premium effects can only be loaded into FX engines 1-8.

- CHANNEL Presets
    - Gain and Phantom power status which are part of the source associated to the channel used to create the preset are saved with the presets, but are not loaded by default when applying the preset to another channel; You will have to enable/select the “conn” setting box (see red arrow below) to ensure gain and phantom power are restored; This will also affect the source to the destination channel..



    - If you set the target FX slot of a Channel Preset (one of the inserts) to NONE, the insert is switched off when loading the Preset.
    - Presets can contain insert effect data; Care must be taken when loading them, as effect engines might be used in other Channels.
    - Channel/Aux/Bus/Main/Matrix Presets can only be loaded into corresponding Channel of course.
    - Bus Presets contain Channel feeds into the bus (FEED scope).

- Old ROUTING Presets can be loaded as Snapshots (scope is set accordingly). For new routing Presets, just use Snapshots and use Scope to limit loading to routing parameters only.

Audio Clips

WING Show control enables using Audio Clips as Scene entities. One can therefore include a reference to a wav file from a USB stick or stored in WING's internal file system as a Scene that can be part of a Show file. Library navigation functions can be used to launch (load) Audio Clips that are part of a Show as they do for any other Scene entity.

Item Tags and arbitrary MIDI data

Scenes can be tagged, allowing easy MIDI and CC button recall. Tags work as follows:

- An item tag can be set to any Library item (Scene, Snip, Clip, Presets or Audio Clip) from the Show tab under LIBRARY.
- Tags starting with #1 .. #128 [to match with MIDI data 0…0x7f] can be recalled with a corresponding MIDI program change on MIDI channel, or with custom control buttons [using the SCENE RECALL setting for said buttons].
- You can use tags up to 16384 with #BANK.PGM, so #2.1 is the same as #129.
- All Library items can be recalled with MIDI program changes (including bank when > 128) on MIDI channel 7 when enabled. As a result, one can address 128 Scenes using tags on channel 8 and all 1000 via bank/program change on channel 7.
- Be aware that Tags are not necessarily in the same order as MIDI program changes are. So recalling items using TAGS on MIDI CH8 and recalling items using program increment on MIDI CH7 May not recall items in the same order.
- A tag can be assigned to several Library items; In that case, WING doesn't check for exclusiveness of tags and the first one found in the list of Library items wins.
- You can send arbitrary MIDI data with each Scene recall (use hexadecimal notation, separator is optional, i.e. *C002* or *B0,01,7F*). This arbitrary MIDI data can be part and save with an empty snippet, enabling a very flexible control of external MIDI devices directly from the console.

Custom Recall Types & Edit Scope[62]

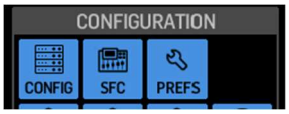
When editing scopes, a list of icons displays on the screen. Most of them are explicit, but some (listed below) will regroup several items or parameters under a single icon that can be selected or un-selected depending on the scope edit or recall operation the user wants to perform. The paragraphs below list the different parameters that are covered by these icons.

[62] Many thanks to Andy Lauer for providing these details.

CONTENTS Scopes (orange Icons)

CUST: Icon / Name / Color / Light on, off
TAGS: Custom Tags / DCA, Mute, Talk Tags
CONN: Source A, B / Main, Alt Status / Input Select Status
(No source Mute or Mono-Stereo-MS)
IN: Trim / Balance / Phase flip
FILTER: LPF / HPF / TILT (Max, AP90, AP180) with all settings
DELAY: On, Off Status / Delay Time
GATE: All settings of the gate (Type / Settings / Side chain …)
DYN: All settings of the dynamic (Type / Settings / Sidechain …)
PRE: Assignment of the FX Plugin (without FX settings)
POST: Assignment of the FX Plugin (without settings), Automix Settings (X, Y, Amount)
EQ: All settings of the EQ (including type of EQ bands), TAP EQ Settings in Bus Sends
PAN: Pan and Width Settings
MAIN: Levels / On, Off Settings / Pre, Post Settings
SEND: Levels / Pan Settings / All status (On, Off / Mode Link / Send Pan / Send Mute / Mode)
FDR: Fader Levels
MUTE: Channel Mutes (no Source Mute)
CONFIG: Process Order / Tap Point / Solo Bus Status

CONFIGURATION Scopes (blue icons)



CONFIG

Monitor Page - Monitor Control: Mute Status, Output Status
Monitor Page - Talkback: – Talk Channel Assign, All Talkback Preferences
Monitor Page – Monitor A and B: All settings and Align options (Delay / EQ / Invert)
Setup – Audio: Main Link, DCA Mutegroups, Startup Main Mute, Automix X - Y (enable status), Solo Mode, Channel Solo, Bus Solo, Main Solo, Matrix Solo, Source Solo
Setup – Surface: Main Meter Dropdown, Main Meter Tap, Show Source On Scribble
SD Card: All settings except "Link Status"
RTA: Range, Decay, Detector, Autogain on/off, Fixed gain value

SFC

Setup – Audio: Mutegroup/SIP Override, Exclusive Solo, Solo Follows Select, Select Follows Solo, BUS/MAIN SoF Activates Solo
Setup – Surface: All The "Lights" settings, Full Fader Paging, Channel Meters, Bus Meters, Main Meters, Matrix Meters, DCA Meters, Screen Follows Ch Strip, Ch Autoselect, User Layer Link, Use F1- F3 As Custom Controls, Right Section Sends On Fader, SOF Buton, Show SOF Frame, Alternative SOF Mode, Sel Dbl Click

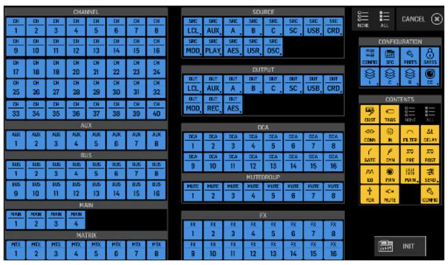
PREFS

Setup – General: Show Meter Page When Locked, Use CRSR/WHEEL For Parameters, Touch Fader Select, Touch Fader Res, Mouse Disables Touch, Mouse Speed value
Setup – Surface: Tap Tempo Flash, Fader Speed
Setup – Remote: Complete Midi Remote Control
Setup – DAW: All DAW Settings

Not Saved in Snapshots:

Monitor Page – Monitor Control: DIM and MONO Buton, TALK A and TALK B Buton
Setup – General: Console Name, Time Date, USB Host Speed, Confirm Library Load, Confirm Library Update
Setup – Audio: Audio Clock (Rate and Sync Source dropdowns), "INPUT SELECT" switch status, Startup Main Mute, Global Input Select Override, USB AUDIO (In/Out dropdown)
Setup-Remote: HA Remote (All settings), Network Settings (dropdown incl. addresses), Remote Lock (OSC / TCP)
4 Track Recorder: Settings (2/4 Ch – 16/24 Bit)
SD-Card: Link Status

INIT Scopes[63]



In the INIT screen, the following settings/parameters will only be initialized/"recalled" if you initialize the desk with **ALL** parameters/settings selected. If anything is taken out of the initialization scope the settings below won't be initialized.

- Clock rate and (sync) source
- Global input select
- USB Audio channel configuration
- Startup main mute
- Global input select override
- HA Remote settings
- The OSC Setting in Setup -> Remote -> Remote Lock
- The DAW control preset.

The following settings will never be initialized/"recalled".

- Console Name
- USB Host Speed
- Clock
- Everything in Setup -> Remote -> Network (IP Adress etc.)
- The TCP Setting in Setup -> Remote -> Remote Lock
- Talk, Headphone and Monitor level (physical knobs on the surface)
- Monitor Mono and Dim (physical buttons on the surface)
- Talk A and Talk B on/off (physical buttons on the surface)
- Everything in the Library

WING Startup Control

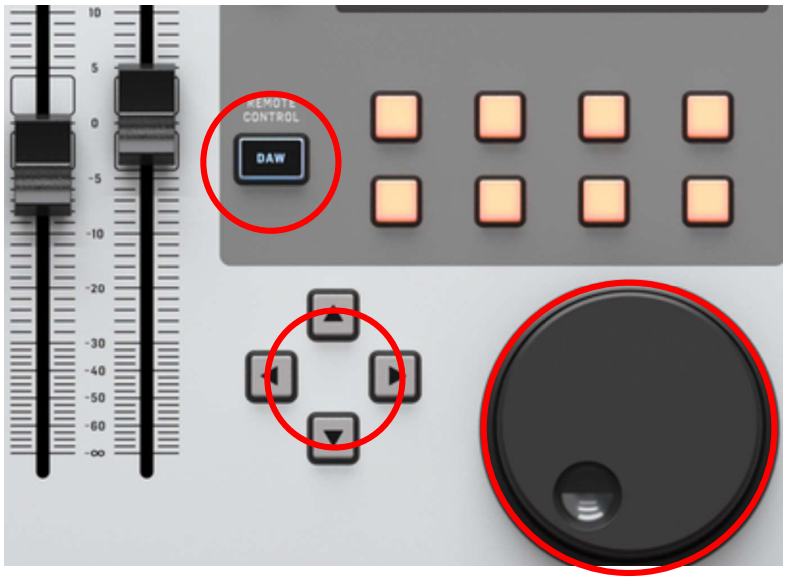
During startup, the console will automatically load a Show, Snapshot or Snippet with the following name when placed in a folder called **STARTUP** in the root of the internal data partition. This can be bypassed when holding the LIBRARY button during power up. Files have to start with the letters "STARTUP", such as

[63] From @sinste from the https://behringer.world forum

- STARTUP.show, or STARTUP_myfile.show for ex.
- STARTUP.snap, or STARTUP_myfile.snap for ex.
- STARTUP.snip, or STARTUP_myfile.snip for ex.

Appendix: MIDI Setup for REAPER Control Surface Use

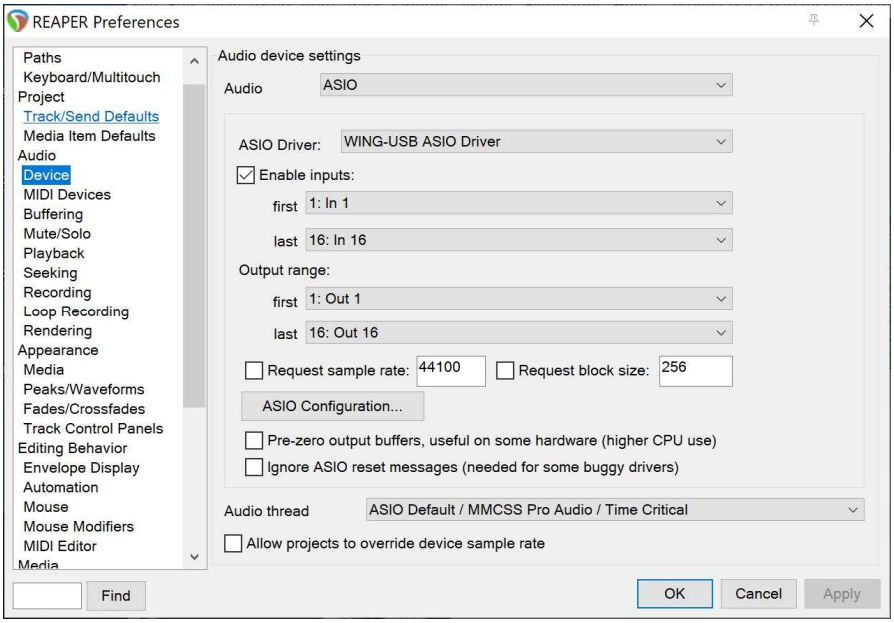
This section is not directly related to programming, but can prove useful when it comes to using WING in a studio, with REAPER™ as a companion DAW software.
The simplest and most complete way to connect all elements together is to use MIDI over USB, MCU mode. This will not only provide a link for REAPER's audio to be sent to WING for audio processing, but will also enable several MIDI channels that can be called for using WING as a control surface and transport controls for REAPER.
In order to achieve this, you will first make sure you have a USB connection between your WING and PC.
You can at any time flip between WING controls and MIDI DAW control using the `DAW Remote Control` button circled in red below and situated left of the group of 8 buttons above the Jog Wheel:



Shown above: the `DAW Remote Control` button, the `Jog wheel`, and the `4 directional keys` mentioned in the coming pages.

REAPER Audio Setup

You then adjust REAPER ASIO interface (and WING setup) to get ASIO channels for audio. The routing on your WING must map USB Inputs to your channel strips. Faders for channel strips should be ideally set to `0dB`. Main strip fader should for the time being be set to `-oo`.
The figure below shows an example for a 16 in/16 out ASIO setup `(Options→Preferences→Audio→Device)`.

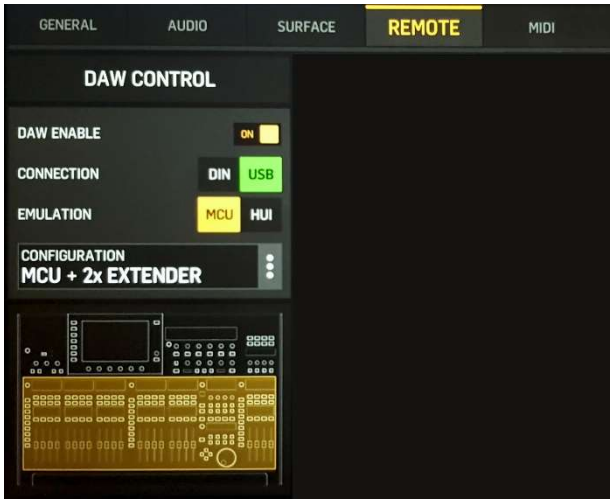


MIDI

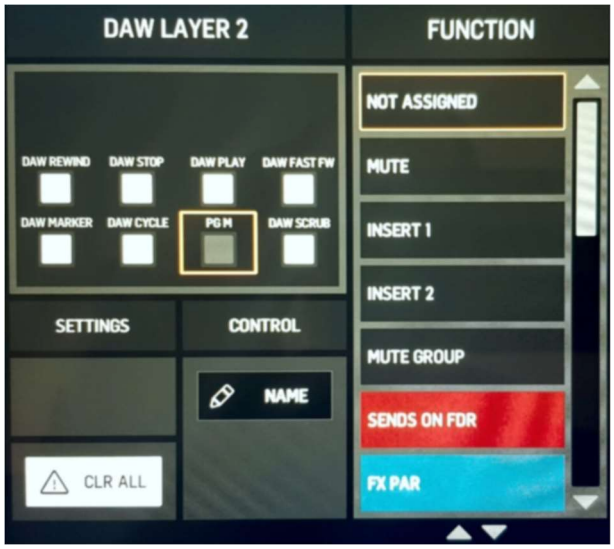
MIDI includes two parts (besides the USB connection mentioned above). The first one is relative to setting WING as a DAW control surface, the second one relates to transport controls.
REAPER DAW control surface is obtained through the `SETUP→Remote` screen. In the left part of the screen, you will choose `USB MIDI` and `MCU+2xExtenders` for a full 24 strips DAW control.

WING MIDI setup

See below the corresponding WING setup screen which can be set from the `SETUP->REMOTE` WING screen. We show here the setup for using a full 24 WING channel strips for MIDI remote control of REAPER, using the MCU + 2 distinct extenders over USB MIDI. You can limit the surface to the controller or controller + 1 extender.



Transport controls proposed here include REW, Fast Forward, Stop/Play/Pause, Scrub, Jog Wheel and more, directly from the lower section of the WING controls. A simple/classic example of implementation is shown below and the setup of these functions is performed after pressing the WING controls' `View` button and assigning keys one by one using the WING main LCD screen.

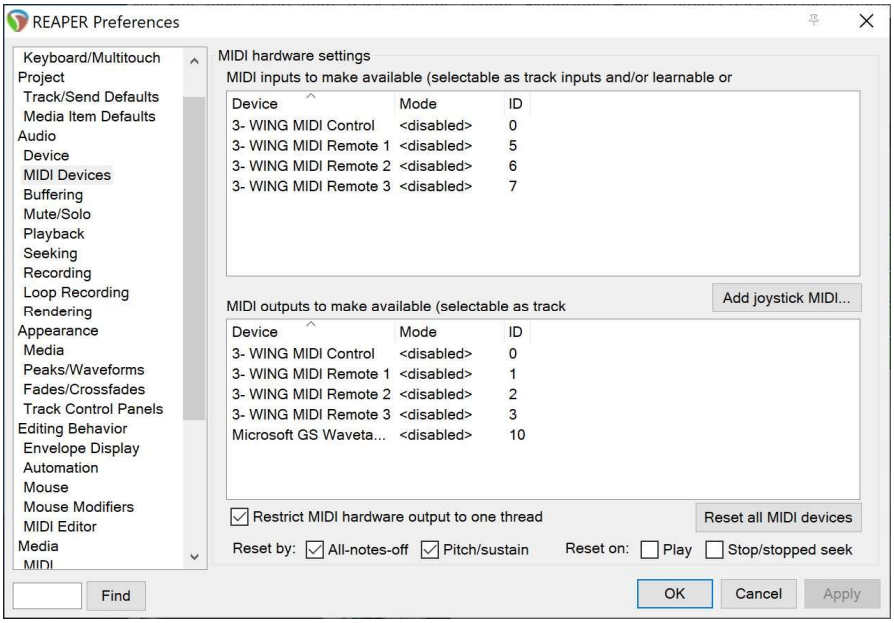


Once modified, the custom transport button layouts can be saved as WING presets. A simple REAPER Control Surface preset is available for download at [https://sites.google.com/site/patrickmaillot/docs/ REAPER DAWsetupV01.snp](https://sites.google.com/site/patrickmaillot/docs/), resulting in the following "DAW LAYER 2" Control Section assignments[64]:

[64] The 3 other layers, and the rest of the console settings, are left untouched

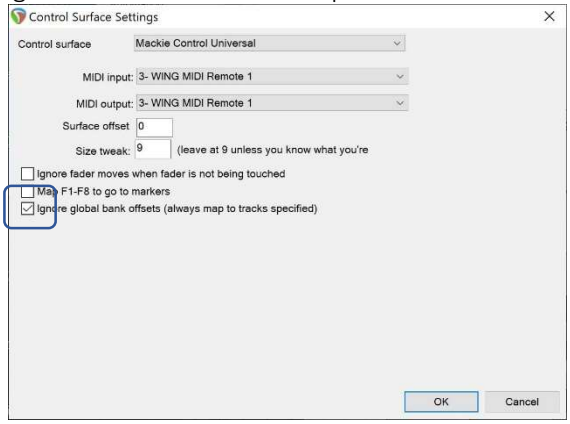
REAPER MIDI setup

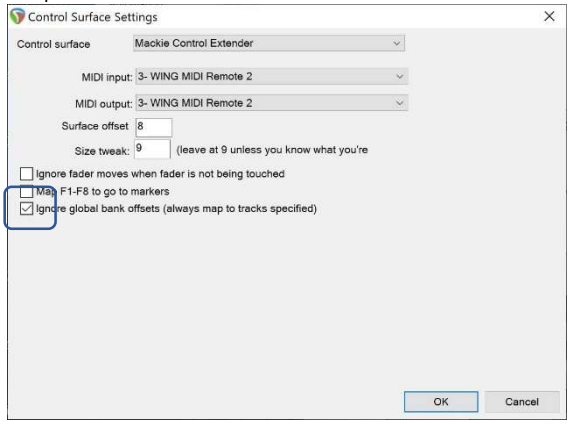
REAPER needs a few simple MIDI settings to correctly enable WING acting as DAW control surface.
When USB is connected, 4 WING MIDI devices appear in the REAPER MIDI devices panel, accessible under `Options→Preferences→Audio→MIDI Devices`. The `MIDI ID` values are managed by REAPER, but can be set as needed, making sure active/enabled device numbers don't duplicate from one active MIDI device to another.
This is shown below:



In the case of DAW control use, all WING MIDI devices above must remain `<disabled>` in the REAPER MIDI Devices panel in order to be used as a control surface communication MIDI device; REAPER will report errors otherwise.

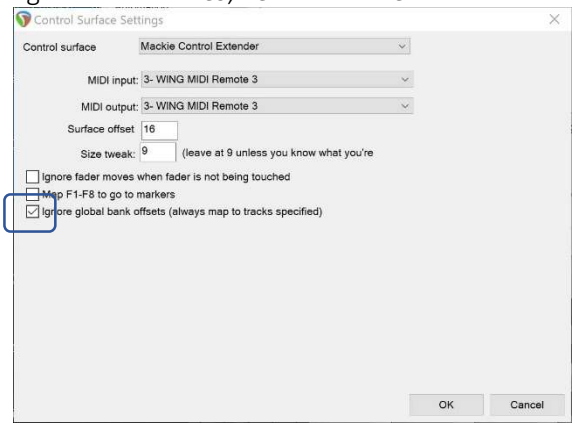
Remember we have setup WING as `USB/MIDI`, `MCU+2xExtenders` in order to cover three times 8 faders, so the full set of channel strips of WING can be used as surface control strips for REAPER.
In the REAPER control Surface panel `(Options→Preferences→Conrol/OSC/Web)`, you will need to add three separate MCU controllers, the first one is a `Mackie Control Universal` device. Controllers 2 and 3 are `Mackie Control Extender` devices. Each device will connect to a `WING MIDI remote device [1, 2, 3]` respectively, ensuring the `surface offset` parameter is set accordingly to its respective WING group of 8 channel strips.
The 4 figures below show an example of REAPER MIDI setup[65].

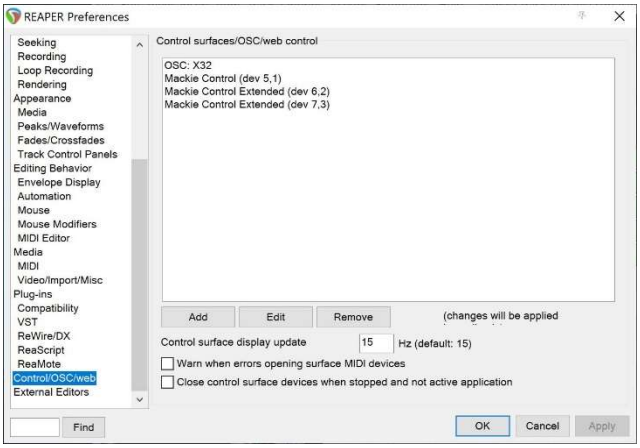




[65] Note that you may have more than one set of WING MIDI remote control, 1, 2, and 3 showing depending on your configuration, or depending on the system state at last reboot or MIDI drivers enable state.

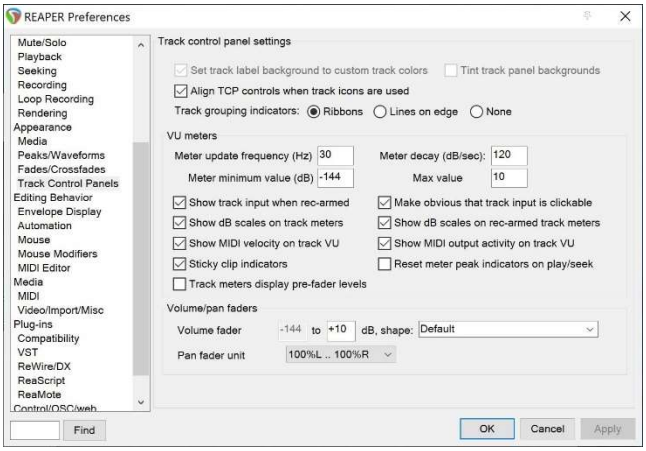
Note the "`Surface offset`" changes as we set `MCU`, `MCE #1` and `MCE #2`





: If the "ignore global bank offsets" flags are **not** checked in the REAPER MIDI surface control setup panels above, using the ◀ 4 and 4 ▶ WING buttons will enable you to navigate left and right in the REAPER tracks if more than 24 REAPER tracks are available.
The current global start index is shown at the top left of the DAW transport scribbles ('01" circled in red below)

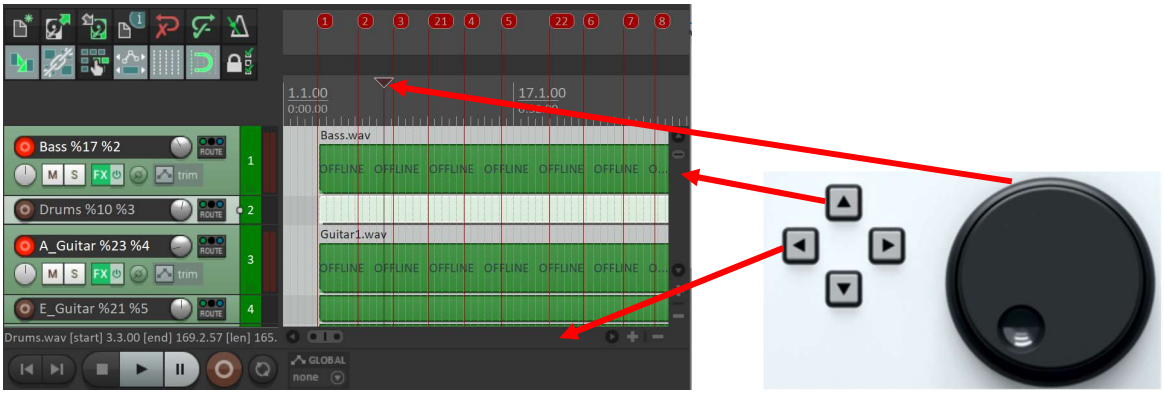
One last setting in REAPER consists in setting the fader scale to values matching WING -144dB→10dB faders. This can be done in the `Options→Preferences→Track Control Settings` panel by setting the min `Volume fader` to `-144dB`, the `max` to `+10dB` and selecting a `shape` type of `Default`, as presented below:



With the settings above, and WING DAW mode setup to `USB MIDI`, `MUC+2xExtenders`, you now have a 24 channel strips DAW surface control to manage REAPER tracks (`Volume, Solo, Select, Mute`) from your WING, and vice-versa (i.e. changes made on a surface (WING or REAPER) will reflect on the other);
REAPER tracks' `Pan` control can be achieved using the 4 rotary knobs in the WING control zone, situated just below the "Custom Controls" silkscreened text.
The `UP` and `DOWN` buttons on the left of the control zone can be used to navigate within REAPER strips as they are mapped to the WING surface strips.

REAPER tracks Rec/arm is possible using the buttons from the lower row of buttons in the WING control zone. The picture below shows Bass and A_Guit being armed for recording:

The `4 directional keys` located at left of the `Jog wheel` will navigate into the REAPER audio window (i.e. identical to moving the audio window elevators).
The WING `Jog wheel` will move the REAPER audio cursor left and right (sometimes with a slight lag), and if the `Play` and `Scrub` buttons are simultaneously active, moving the wheel will scrub through audio after a small timeout not moving the wheel.



Flip to the WING audio standard controls and move the WING Main fader(s) to get some audio out, remember all other active USB inputs should be set at `0dB`.
Flipping back to `DAW Remote` mode and press the `Play` button. REAPER will start playing audio; the audio signal will flow to WING using USB/ASIO drivers, and will be managed by the WING audio engines. The faders on WING (in control surface mode) act as remote controls to REAPER track faders (and vice-versa) using USB MIDI.
All set!

Important note on USB & MIDI: Changing the clock rate or number of USB Audio channels on WING causes USB to disconnect for a few seconds (including MIDI). On certain operating systems, this may also reset already active MIDI connections. This could happen when loading snapshots with different clock rate or USB Audio configuration.

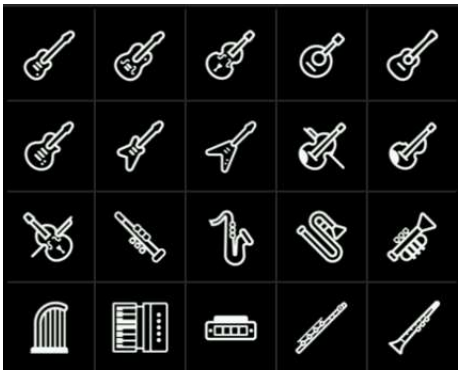
Appendix: WING Icons

The table below gives the list of icons available with WING. Icon number ranges are listed to the right of the icons.

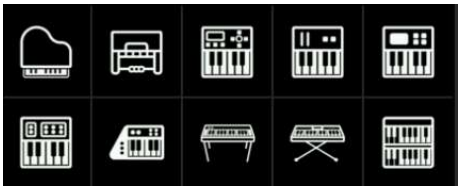
General:
[0…14]

Vocals and Mics:
[100…114]

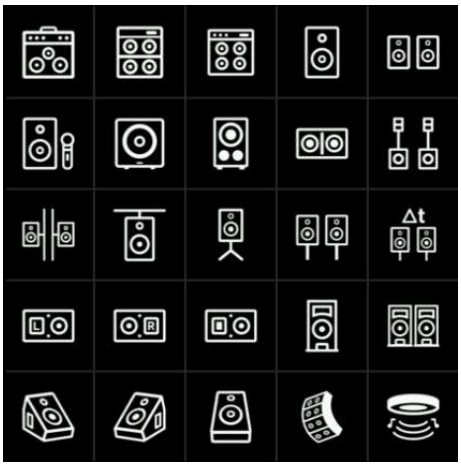
Drums and Percussions:
[200…224]



Strings and Winds:
[300…319]



Keys:
[400…409]

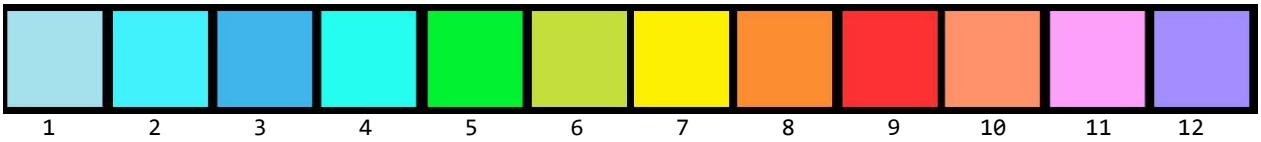


Speakers:
[500…524]

Specials:
[600…614]

Appendix: WING Colors

WING colors are used in several areas such as channel strip color, scribble color, etc. The known colors are shown below and indexed as values 1 to 12:



1	gray blue
2	medium blue
3	dark blue
4	turquoise
5	green
6	olive green
7	yellow
8	orange
9	red
10	coral
11	pink
12	mauve

Appendix: WING GPIOs:

The WING digital mixing console is offering 4 GPIOs (General Purpose Input/Output) which can be very useful in the studio or live situations. This paragraph shows how to use them in different modes. Let's look at what GPIOs can offer.

At the rear of the console, two TRS jack sockets provide connections to 4 GPIOS. Each of the TRS sockets is depicted below. Lug L3 is common to the 2 GPIOs supported by each socket. Lugs L1 and L2 are respectively used for GPIO 1 or A, 2 or B or 3 or C, 4 or D, depending on the socket used.

WING GPIO 'mode' settings can be any of the following: `TGLNO, TGLNC, INNO, INNC, OUTNO, OUTNC`. These are represented by OSC patterns `/$ctl/gpio/1..4/mode`, and correspond to:

`TGLNO`	Toggle, Normally Opened
`TGLNC`	Toggle, Normally Closed
`INNO`	Input, Normally Opened
`INNC`	Input, Normally Closed
`OUTNO`	Output, Normally Opened
`OUTNC`	Output, Normally Closed

WING GPIO 'state' values can be 0 for `open/OFF` (light off), or 1 for `close/ON` (light on). These correspond to OSC patterns `/$ctl/gpio/1..4/gpstate`.

Electrical connections:

- `INNO / INNC`: The console provides approx. 5V between A/B/C/D and Common. The application of a short, dropping voltage to 0V will change the state of the respective GPIO between `open` and `close`, depending on the `NO/NC` mode.

- `OUTNO / OUTNC`: The console provides approx. 5V between A/B/C/D and Common; The voltage presented by the console goes from near 5V to 0V depending on the state (`open` or `close`) and the `NO/NC` mode of the respective GPIO.

- `TGLNO / TGLNC`: This is to toggle the internal state of the GPIO. The console provides approx.. 5V between A/B/C/D and Common; changing the state of the respective GPIO does not change the voltage provided by the console.

Appendix: MCU [DAW BUTTONS] commands list

OSC	MCU action	MIDI (port 4)		OSC	MCU action	MIDI (port 4)
T1	STOP	90, 5D, 7F/00		V7	BUSES (VIEW)	90, 43, 7F/00
T2	PLAY	90, 5E, 7F/00		V8	OUTPUTS (VIEW)	90, 44, 7F/00
T3	RECORD	90, 5F, 7F/00		V9	USER (VIEW)	90, 45, 7F/00
T4	REWIND	90, 5B, 7F/00		V10	MIX (VIEW)	
T5	FAST FWD	90, 5C, 7F/00		V11	EDIT (VIEW)	
T6	MARKER	90, 54, 7F/00		V12	TRANSPORT (VIEW)	
T7	NUDGE	90, 55, 7F/00		V13	MEM/LOC (VIEW)	
T8	CYCLE	90, 56, 7F/00		V14	STATUS (VIEW)	
T9	DROP	90, 57, 7F/00		V15	ALT (VIEW)	
T10	REPLACE	90, 58, 7F/00		AU1	READ/OFF (AUTOM)	90, 4A, 7F/00
T11	SCRUB	90, 65, 7F/00		AU2	WRITE (AUTOM)	90, 4B, 7F/00
T12	SHUTTLE			AU3	TRIM (AUTOM)	90, 4C, 7F/00
T13	RETURN TO ZERO			AU4	TOUCH (AUTOM)	90, 4D, 7F/00
T14	GO TO END			AU5	LATCH (AUTOM)	90, 4E, 7F/00
T15	IN			AU6	OFF (AUTOM)	
T16	OUT			AU7	FADER (AUTOM)	
T17	PRE			AU8	PAN (AUTOM)	
T18	POST			AU9	MUTE (AUTOM)	
T19	ONLINE			AU10	SEND (AUTOM)	
T20	QUICK PUNCH			AU11	SEND MUTE (AUTOM)	
N1	UP (NAV)	90, 60, 7F/00		AU12	PLUG-IN (AUTOM)	
N2	DOWN (NAV)	90, 61, 7F/00		SY1	SHIFT	90, 46, 7F/00
N3	LEFT (NAV)	90, 62, 7F/00		SY2	OPTION	90, 47, 7F/00
N4	RIGHT (NAV)	90, 63, 7F/00		SY3	CTRL	90, 48, 7F/00
N5	ZOOM	90, 64, 7F/00		SY4	ALT	90, 49, 7F/00
N6	BK <	90, 2E, 7F/00		SY5	SAVE	90, 50, 7F/00
N7	BK >	90, 2F, 7F/00		SY6	UNDO	90, 51, 7F/00
N8	CH <	90, 30, 7F/00		SY7	CANCEL	90, 52, 7F/00
N9	CH >	90, 31, 7F/00		SY8	ENTER	90, 53, 7F/00
A1	TRACK (ASSIGN)	90, 28, 7F/00		SY9	EDIT MODE	
A2	SEND (ASSIGN)	90, 29, 7F/00		SY10	EDIT TOOL	
A3	PAN (ASSIGN)	90, 2A, 7F/00		OT1	FLIP	90, 32, 7F/00
A4	PLUG-IN (ASSIGN)	90, 2B, 7F/00		OT2	GROUP	90, 4F, 7F/00
A5	EQ (ASSIGN)	90, 2C, 7F/00		OT3	NAME/VALUE	90, 34, 7F/00
A6	INST (ASSIGN)	90, 2D, 7F/00		OT4	TIME/BEATS	90, 35, 7F/00
A7	SEND A (ASSIGN)			OT5	CLICK	90, 59, 7F/00
A8	SEND B (ASSIGN)			OT6	SOLO	90, 5A, 7F/00
A9	SEND C (ASSIGN)			OT7	FOOTSW A	90, 66, 7F/00
A10	SEND D (ASSIGN)			OT8	FOOTSW B	90, 67, 7F/00
A11	SEND E (ASSIGN)			OT9	DEFAULT	
A12	INPUT (ASSIGN)			OT10	SUSPEND	
A13	OUTPUT (ASSIGN)			OT11	BYPASS	
A14	ASSIGN (ASSIGN)			OT12	RECRDY ALL	
A15	SHIFT (ASSIGN)			E1	CUT (EDIT)	
A16	MUTE (ASSIGN)			E2	COPY (EDIT)	
F1	F1	90, 36, 7F/00		E3	PASTE (EDIT)	
F2	F2	90, 37, 7F/00		E4	SEPARATE (EDIT)	
F3	F3	90, 38, 7F/00		E5	CAPTURE (EDIT)	
F4	F4	90, 39, 7F/00		E6	DELETE (EDIT)	
F5	F5	90, 3A, 7F/00		E7	ASSIGN (EDIT)	
F6	F6	90, 3B, 7F/00		E8	COMPARE (EDIT)	
F7	F7	90, 3C, 7F/00		E9	BYPASS (EDIT)	
F8	F8	90, 3D, 7F/00		E10	INS/PARAM (EDIT)	
V1	GLOBAL (VIEW)	90, 33, 7F/00		SP1	FADER TOUCH [MUTE]	
V2	MIDI (VIEW)	90, 3E, 7F/00		SP2	V-POT CTRL [SEL/SOLO]	
V3	INPUTS (VIEW)	90, 3F, 7F/00		SP3	RECRDY CTRL [SEL]	
V4	AUDIO TRACKS (VIEW)	90, 40, 7F/00		SP4	AUTO [SEL]	
V5	INSTRUMENT (VIEW)	90, 41, 7F/00		SP5	V-SEL [SEL]	
V6	AUX (VIEW)	90, 42, 7F/00		SP6	INSERT [SEL]	

Appendix: MCU [DAW V-POTS] commands list

OSC	MCU action	MIDI (port 4)		OSC	MCU action	MIDI (port 4)
M1P	V-POT M1 Push	90, 20, 7F/00		M1	V-POT M1	B0, 10, 01/41
M2P	V-POT M2 Push	90, 21, 7F/00		M2	V-POT M2	B0, 11, 01/41
M3P	V-POT M3 Push	90, 22, 7F/00		M3	V-POT M3	B0, 12, 01/41
M4P	V-POT M4 Push	90, 23, 7F/00		M4	V-POT M4	B0, 13, 01/41
M5P	V-POT M5 Push	90, 24, 7F/00		M5	V-POT M5	B0, 14, 01/41
M6P	V-POT M6 Push	90, 25, 7F/00		M6	V-POT M6	B0, 15, 01/41
M7P	V-POT M7 Push	90, 26, 7F/00		M7	V-POT M7	B0, 16, 01/41
M8P	V-POT M8 Push	90, 27, 7F/00		M8	V-POT M8	B0, 17, 01/41
E1P	V-POT EXT1 Push			E1	V-POT EXT1	
E2P	V-POT EXT2 Push			E2	V-POT EXT2	
E3P	V-POT EXT3 Push			E3	V-POT EXT3	
E4P	V-POT EXT4 Push			E4	V-POT EXT4	
E5P	V-POT EXT5 Push			E5	V-POT EXT5	
E6P	V-POT EXT6 Push			E6	V-POT EXT6	
E7P	V-POT EXT7 Push			E7	V-POT EXT7	
E8P	V-POT EXT8 Push			E8	V-POT EXT8	
E9P	V-POT EXT9 Push			E9	V-POT EXT9	
E10P	V-POT EXT10 Push			E10	V-POT EXT10	
E11P	V-POT EXT11 Push			E11	V-POT EXT11	
E12P	V-POT EXT12 Push			E12	V-POT EXT12	
E13P	V-POT EXT13 Push			E13	V-POT EXT13	
E14P	V-POT EXT14 Push			E14	V-POT EXT14	
E15P	V-POT EXT15 Push			E15	V-POT EXT15	
E16P	V-POT EXT16 Push			E16	V-POT EXT16	
				JOG	JOG WHEEL	B0, 3C, 01/41

Appendix: MCU [DAW REMOTE MCU] commands list

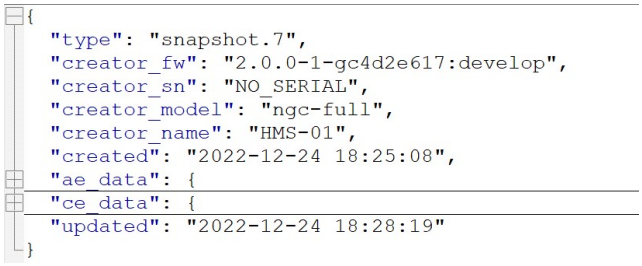
OSC	MCU action	MIDI (port 4)
M1	V-POT M1	B0, 10, 01/41
M2	V-POT M2	B0, 11, 01/41
M3	V-POT M3	B0, 12, 01/41
M4	V-POT M4	B0, 13, 01/41
M5	V-POT M5	B0, 14, 01/41
M6	V-POT M6	B0, 15, 01/41
M7	V-POT M7	B0, 16, 01/41
M8	V-POT M8	B0, 17, 01/41
E1	V-POT EXT1	
E2	V-POT EXT2	
E3	V-POT EXT3	
E4	V-POT EXT4	
E5	V-POT EXT5	
E6	V-POT EXT6	
E7	V-POT EXT7	
E8	V-POT EXT8	
E9	V-POT EXT9	
E10	V-POT EXT10	
E11	V-POT EXT11	
E12	V-POT EXT12	
E13	V-POT EXT13	
E14	V-POT EXT14	
E15	V-POT EXT15	
E16	V-POT EXT16	
JOG	JOG WHEEL	B0, 3C, 01/41

Appendix: WING Snapshot and JSON Data Structure:

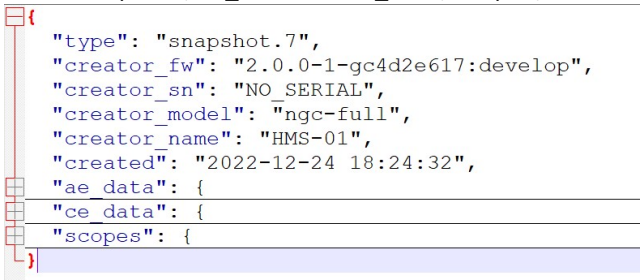
A WING snapshot file (also called Snapfile when saved to a file) is organized as a collection of classes, sub-classes and objects regrouping attributes and values in logical groups. These can be represented as a hierarchical tree. A `JSON`[66] notation is used to describe and store the hierarchical tree.
A complete WING snapfile is close to 460000 bytes and 28800 lines, containing a rather complex hierarchical list of object identifiers and their associated values.
A WING snapfile does not contain read-only objects; i.e. there are more elements available than the one saved in a snapfile!

Global Snapfile

A snapfile is divided in either:
3 sections: `description`, `ae_data` and `ce_data`, as shown below:



4 sections: `description`, `ae_data` and `ce_data`, `scopes,` as shown below:



Description

description: This small section contains (as its name suggest) a description for the snapshot, including name, and elements corresponding to the WING that generated the snapshot. “created” lists the date and time of the creation of the snapfile, while “updated” will retain the date and time of the most recent update made to the file.

“type”: string,
“creator_fw”: string,
“creator_sn”: string,
“creator_model”: string,
“creator_name”: string,
“created”: date time,
“updated”: date time,

[66] JavaScript Object Notation: an efficient way to represent structured objects. Also used as a data-interchange format.

scopes

scopes: A large set of *BooLean* {**'+',' '**} values to list what has been 'marked' at snapshot time. This can be used as a reminder of the initial purpose of the snapshot.
The scopes class contains the following objects:

ch, aux, bus, main, mtx, dca, mute, fx, source, output, area, custom, setup, mainflt, sendflt;

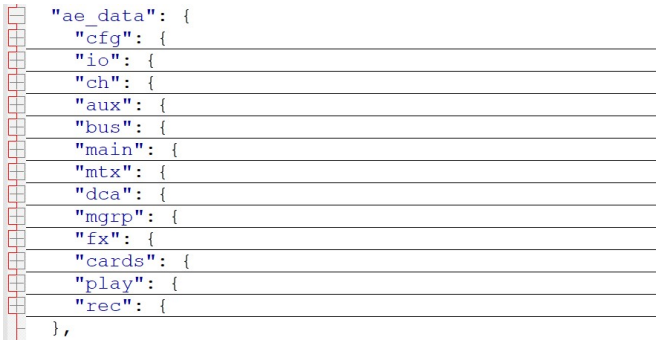
For example:

```
"scopes": {
  "ch": "++++++++++++++++++++++++++++++++++++",
  "aux": "++++++++",
  "bus": "++++++++++++++++",
  "main": "++++",
  "mtx": "++++++++",
  "dca": "++++++++++++++++",
  "mute": "++++++++",
  "fx": "++++++++++++++++",
  "source": {
    "LCL": "++++++++",
    "AUX": "++++++++",
    "A": "++++++++++++++++++++++++++++++++++++++++++++",
    "B": "++++++++++++++++++++++++++++++++++++++++++++",
    "C": "++++++++++++++++++++++++++++++++++++++++++++",
    "SC": "++++++++++++++++++++++++++++++++",
    "USB": "++++++++++++++++++++++++++++++++++++++++++++",
    "CRD": "++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++",
    "MOD": "++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++",
    "PLAY": "++++",
    "AES": "++",
    "USR": "++++++++++++++++++++++",
    "OSC": "++"
  },
  "output": {
    "LCL": "++++++++",
    "AUX": "++++++++",
    "A": "++++++++++++++++++++++++++++++++++++++++++++",
    "B": "++++++++++++++++++++++++++++++++++++++++++++",
    "C": "++++++++++++++++++++++++++++++++++++++++++++",
    "SC": "++++++++++++++++++++++++++++++++",
    "USB": "++++++++++++++++++++++++++++++++++++++++++++",
    "CRD": "++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++",
    "MOD": "++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++",
    "REC": "++++",
    "AES": "++"
  },
  "area": {
    "L": "+++++++",
    "C": "++++++",
    "R": "+++++++"
  },
  "custom": "++++++++++++++++++++++",
  "setup": "+++",
  "filters": "              ",
  "mainflt": "    ",
  "sendflt": "                        "
}
```

Scopes are not elements that can be programmatically changed. They are only set at snapshot time using the console main LCD. As mentioned above, they are optionally saved at save time to notify what was targeted for save/update.

ae_data

ae_data stands for "Audio Engine", and regroups a rather large set of attributes and values aimed at registering all main settings of the WING audio engine, such as Routing, Channel EQ settings, FX parameter values, etc., as shown in the figure below:



In the next pages, we present the structure, 1 block of parameters at a time. Understanding what parameters are present in each block is a good way to better grasp and understand the vast range of capabilities WING offers. It is also a good way to envision the parameter list one can get and set using **wapi** (described earlier in this document) as the JSON structure parameters matches the tokens used by the API for **wapi** `get()` and `set()` functions.

Indeed, all tokens related to the audio engine can be directly coded from the JSON description, for example, the C-like token notation for the JSON *cfg.mon.1.pan* element is named CFG_MON_1_PAN.

We show in the following pages, the contents of the JSON tree structure after a console reset, so default values are listed. In order to reduce the number of pages the JSON structure description would take; the following notation is used:

"abc": {}, means that *"abc"* uses the same structure definition as the previous member in the JSON file, and:

"2":{}…"n": {}, means that objects *"2"* to *"n"* use the same structure definition as the previous member in the JSON file.

```
"ae_data": {
        "cfg": {
            "clkrate": 48000,
            "clksrc": "INT",
            "mainlink": "OFF",
            "dcamgrp": true,
            "muteovr": true,
            "startmute": false,
            "usbacfg": "48/48",
            "sccfg": "AUTO",
            "mon": {
                "1": {
                    "inv": false,
                    "pan": 0,
                    "wid": 100,
                    "eq": {
                        "on": false,
                        "lsg": 0,
                        "lsf": 60.13884,
                        "1g": 0,
                        "1f": 129.8763,
```

```
                "1q": 1.995882,
                "2g": 0,
                "2f": 299.2472,
                "2q": 1.995882,
                "3g": 0,
                "3f": 699.4875,
                "3q": 1.995882,
                "4g": 0,
                "4f": 1499.788,
                "4q": 1.995882,
                "5g": 0,
                "5f": 2992.471,
                "5q": 1.995882,
                "6g": 0,
                "6f": 6013.884,
                "6q": 1.995882,
                "hsg": 0,
                "hsf": 11994.42
            },
            "lim": 0,
            "dly": {
                "on": false,
                "m": 0.1
            },
            "dim": 20,
            "pfldim": 12,
            "eqbdtrim": 0,
            "srclvl": 0,
            "srcmix": -144,
            "src": "MAIN.1",
            "tags": ""
        },
        "2": {}
    },
    "solo": {
        "mode": "LIVE",
        "mon": "A",
        "mute": false,
        "chtap": "PFL",
        "bustap": "AFL",
        "maintap": "PFL",
        "mtxtap": "PFL",
        "srcsolo": "OFF"
    },
    "rta": {
        "rtasrc": 0,
        "rtatap": "IN",
        "rtadecay": "MED",
        "rtadet": "PEAK",
        "rtarange": 30,
        "rtagain": 0,
        "rtaauto": true
    },
    "mtr": {
        "scopesrc": 0,
        "scopetap": "IN"
    },
    "talk": {
        "assign": "OFF",
        "indiv": false,
        "A": {
            "mode": "AUTO",
            "mondim": false,
            "busdim": 0,
            "B1": false,
            "B2": false,
            "B3": false,
            "B4": false,
            "B5": false,
            "B6": false,
            "B7": false,
            "B8": false,
            "B9": false,
            "B10": false,
            "B11": false,
            "B12": false,
            "B13": false,
```

```
                "B14": false,
                "B15": false,
                "B16": false,
                "M1": false,
                "M2": false,
                "M3": false,
                "M4": false
            },
            "B": {}
        },
        "harmt": {
            "a": false,
            "b": false,
            "c": false
        },
        "custsync": {
            "a": false,
            "b": false,
            "c": false
        },
        "amix": {
            "x": true,
            "y": true
        }
    },
    "io": {
        "altsw": false,
        "in": {
            "LCL": {
                "1": {
                    "mode": "M",
                    "g": 0,
                    "vph": false,
                    "mute": false,
                    "pol": false,
                    "col": 1,
                    "name": "",
                    "icon": 1,
                    "tags": "",
                    "rmt": "OFF",
                    "rcvc": false
                },
                "2": {},..."8": {}
            },
            "AUX": {
                "1": {
                    "mode": "M",
                    "mute": false,
                    "pol": false,
                    "col": 1,
                    "name": "",
                    "icon": 2,
                    "tags": ""
                },
                "2": {},..."8": {}
            },
            "A": {
                "1": {
                    "mode": "M",
                    "g": 0,
                    "vph": false,
                    "mute": false,
                    "pol": false,
                    "col": 1,
                    "name": "",
                    "icon": 0,
                    "tags": "",
                    "rmt": "OFF",
                    "rcvc": false
                },
                "2": {},..."48": {}
            },
            "B": {},
            "C": {},
            "SC": {
                "1": {
                    "mode": "M",
```

```
            "mute": false,
            "pol": false,
            "col": 1,
            "name": "",
            "icon": 0,
            "tags": ""
        },
        "2": {},..."32": {}
    },
    "USB": {
        "1": {
            "mode": "ST",
            "mute": false,
            "pol": false,
            "col": 8,
            "name": "USB 1/2",
            "icon": 605,
            "tags": ""
        },
        "2": {},..."48": {}
    },
    "CRD": {
        "1": {
            "mode": "M",
            "mute": false,
            "pol": false,
            "col": 1,
            "name": "",
            "icon": 0,
            "tags": ""
        },
        "2": {},..."64": {}
    },
    "MOD": {
        "1": {
            "mode": "M",
            "mute": false,
            "pol": false,
            "col": 1,
            "name": "",
            "icon": 0,
            "tags": ""
        },
        "2": {},..."64": {}
    },
    "PLAY": {
        "1": {
            "mode": "ST",
            "mute": false,
            "pol": false,
            "col": 8,
            "name": "2TR",
            "icon": 608,
            "tags": ""
        },
        "2": {},..."4": {}
    },
    "AES": {
        "1": {
            "mode": "M",
            "mute": false,
            "pol": false,
            "col": 1,
            "name": "",
            "icon": 0,
            "tags": ""
        },
        "2": {}
    },
    "USR": {
        "1": {
            "mode": "M",
            "mute": false,
            "pol": false,
            "col": 1,
            "name": "",
            "icon": 0,
```

```
                "tags": "",
                "user": {
                    "grp": "OFF",
                    "in": 1,
                    "tap": "PRE",
                    "lr": "L+R"
                }
            },
            "2": {},..."24": {}
        },
        "OSC": {
            "1": {
                "mode": "M",
                "mute": false,
                "col": 1,
                "name": "",
                "icon": 0,
                "tags": "",
                "osc": {
                    "lvl": -6,
                    "mode": "SINE",
                    "f": 999.992
                }
            },
            "2": {}
            }
        }
    },
    "out": {
        "LCL": {
            "1": {
                "grp": "BUS",
                "in": 1
            },
            "2": {},..."8": {}
        },
        "AUX": {
            "1": {
                "grp": "OFF",
                "in": 1
            },
            "2": {},..."8": {}
        },
        "A": {
            "1": {
                "grp": "OFF",
                "in": 1
            },
            "2": {},..."48": {}
        },
        "B": {},
        "C": {},
        "SC": {
            "1": {
                "grp": "OFF",
                "in": 1
            },
            "2": {},..."32": {}
        },
        "USB": {
            "1": {
                "grp": "OFF",
                "in": 1
            },
            "2": {},..."48": {}
        },
        "CRD": {
            "1": {
                "grp": "OFF",
                "in": 1
            },
            "2": {},..."64": {}
        },
        "MOD": {
            "1": {
                "grp": "OFF",
                "in": 1
```

```
            },
            "2": {},..."64": {}
        },
        "REC": {
            "1": {
                "grp": "OFF",
                "in": 1
            },
            "2": {},..."4": {}
        },
        "AES": {
            "1": {
                "grp": "OFF",
                "in": 1
            },
            "2": {}
        }
    }
},
"ch": {
    "1": {
        "in": {
            "set": {
                "srcauto": false,
                "altsrc": false,
                "inv": false,
                "trim": 0,
                "bal": 0,
                "dlymode": "M",
                "dly": 0,
                "dlyon": false
            },
            "conn": {
                "grp": "LCL",
                "in": 1,
                "altgrp": "OFF",
                "altin": 1
            }
        },
        "flt": {
            "lc": false,
            "lcf": 100.2375,
            "hc": false,
            "hcf": 10018.26,
            "tf": false,
            "mdl": "TILT",
            "tilt": 0
        },
        "clink": true,
        "col": 1,
        "name": "",
        "icon": 1,
        "led": true,
        "mute": false,
        "fdr": -144,
        "pan": 0,
        "wid": 100,
        "solosafe": false,
        "mon": "A",
        "proc": "GEDI",
        "ptap": "5",
        "peq": {
            "on": false,
            "1g": 0,
            "1f": 99.68543,
            "1q": 0.99797,
            "2g": 0,
            "2f": 999.2505,
            "2q": 0.99797,
            "3g": 0,
            "3f": 10016.53,
            "3q": 0.99797
        },
        "gate": {
            "on": false,
            "mdl": "GATE",
            "thr": -40,
```

```
    "range": 40,
    "att": 10,
    "hld": 10,
    "rel": 199.4043,
    "acc": 0,
    "ratio": "1:3"
},
"gatesc": {
    "type": "OFF",
    "f": 1002.374,
    "q": 1.995882,
    "src": "SELF",
    "tap": "IN"
},
"eq": {
    "on": false,
    "mdl": "STD",
    "mix": 100,
    "lg": 0,
    "lf": 80.19642,
    "lq": 0.99797,
    "leq": "SHV",
    "1g": 0,
    "1f": 200,
    "1q": 0.99797,
    "2g": 0,
    "2f": 601.3884,
    "2q": 0.99797,
    "3g": 0,
    "3f": 1499.788,
    "3q": 0.99797,
    "4g": 0,
    "4f": 3990.524,
    "4q": 0.99797,
    "hg": 0,
    "hf": 11994.42,
    "hq": 0.99797,
    "heq": "SHV"
},
"dyn": {
    "on": false,
    "mdl": "COMP",
    "mix": 100,
    "gain": 0,
    "thr": -10,
    "ratio": 3,
    "knee": 3,
    "det": "RMS",
    "att": 50,
    "hld": 20,
    "rel": 152.5652,
    "env": "LOG",
    "auto": true
},
"dynxo": {
    "depth": 6,
    "type": "OFF",
    "f": 1002.374
},
"dynsc": {
    "type": "OFF",
    "f": 1002.374,
    "q": 1.995882,
    "src": "SELF",
    "tap": "IN"
},
"preins": {
    "on": false,
    "ins": "NONE"
},
"main": {
    "1": {
        "on": true,
        "lvl": 0,
        "pre": false
    },
    "2": {},..."4": {}
```

```
            },
            "send": {
                "1": {
                    "on": false,
                    "lvl": -144,
                    "pon": false,
                    "ind": false,
                    "mode": "PRE",
                    "plink": false,
                    "pan": 0,
                    "wid": 100
                },
                "2": {},..."16": {}
            },
            "postins": {
                "on": false,
                "mode": "FX",
                "ins": "NONE",
                "w": 0
            },
            "tags": ""
        },
        "2": {},..."40": {}
    },
    "aux": {
        "1": {
            "in": {
                "set": {
                    "srcauto": false,
                    "altsrc": false,
                    "inv": false,
                    "trim": 0,
                    "bal": 0
                },
                "conn": {
                    "grp": "USB",
                    "in": 1,
                    "altgrp": "OFF",
                    "altin": 1
                }
            },
            "clink": true,
            "col": 8,
            "name": "USB 1/2",
            "icon": 605,
            "led": true,
            "mute": false,
            "fdr": -144,
            "pan": 0,
            "wid": 100,
            "solosafe": false,
            "mon": "A",
            "eq": {
                "on": false,
                "mdl": "STD",
                "mix": 100,
                "lg": 0,
                "lf": 80.19642,
                "lq": 0.99797,
                "leq": "SHV",
                "1g": 0,
                "1f": 200,
                "1q": 0.99797,
                "2g": 0,
                "2f": 601.3884,
                "2q": 0.99797,
                "3g": 0,
                "3f": 1499.788,
                "3q": 0.99797,
                "4g": 0,
                "4f": 3990.524,
                "4q": 0.99797,
                "hg": 0,
                "hf": 11994.42,
                "hq": 0.99797,
                "heq": "SHV"
            },
```

```
            "dyn": {
                "on": false,
                "thr": -36,
                "depth": 12,
                "fast": false,
                "peak": false,
                "ingain": 40,
                "cpeak": 0,
                "cmode": "COMP"
            },
            "preins": {
                "on": false,
                "ins": "NONE"
            },
            "main": {
                "1": {
                    "on": true,
                    "lvl": 0,
                    "pre": false
                },
                "2": {},..."4": {}
            },
            "send": {
                "1": {
                    "on": false,
                    "lvl": -144,
                    "pon": false,
                    "ind": false,
                    "mode": "PRE",
                    "plink": false,
                    "pan": 0,
                    "wid": 100
                },
                "2": {},..."16": {}
            },
            "tags": ""
        },
        "2": {},..."8": {}
    },
    "bus": {
        "1": {
            "in": {
                "set": {
                    "inv": false,
                    "trim": 0,
                    "bal": 0
                }
            },
            "col": 1,
            "name": "",
            "icon": 0,
            "led": true,
            "busmono": false,
            "mute": false,
            "fdr": -144,
            "pan": 0,
            "wid": 100,
            "mon": "A",
            "busmode": "PRE",
            "eq": {
                "on": false,
                "mdl": "STD",
                "mix": 100,
                "lg": 0,
                "lf": 60.13884,
                "lq": 0.99797,
                "leq": "SHV",
                "1g": 0,
                "1f": 129.8763,
                "1q": 0.99797,
                "2g": 0,
                "2f": 299.2472,
                "2q": 0.99797,
                "3g": 0,
                "3f": 699.4875,
                "3q": 0.99797,
                "4g": 0,
```

```
                "4f": 1499.788,
                "4q": 0.99797,
                "5g": 0,
                "5f": 2992.471,
                "5q": 0.99797,
                "6g": 0,
                "6f": 6013.884,
                "6q": 0.99797,
                "hg": 0,
                "hf": 11994.42,
                "hq": 0.99797,
                "heq": "SHV",
                "tilt": 0
            },
            "dyn": {
                "on": false,
                "mdl": "COMP",
                "mix": 100,
                "gain": 0,
                "thr": -10,
                "ratio": 3,
                "knee": 3,
                "det": "RMS",
                "att": 50,
                "hld": 20,
                "rel": 152.5652,
                "env": "LOG",
                "auto": true
            },
            "dynxo": {
                "depth": 6,
                "type": "OFF",
                "f": 1002.374
            },
            "dynsc": {
                "type": "OFF",
                "f": 1002.374,
                "q": 1.995882,
                "src": "SELF",
                "tap": "BUS"
            },
            "preins": {
                "on": false,
                "ins": "NONE"
            },
            "main": {
                "1": {
                    "on": false,
                    "lvl": 0,
                    "pre": false
                },
                "2": {},..."4": {}
            },
            "send": {
                "1": {
                    "on": false,
                    "lvl": -144,
                    "pre": false
                },
                "2": {},..."8": {},
                "MX1": {
                    "on": false,
                    "lvl": -144,
                    "pre": false
                },
                "MX2": {},..."MX8": {}
            },
            "postins": {
                "on": false,
                "ins": "NONE"
            },
            "tags": ""
        },
        "2": {},..."16": {}
    },
    "main": {
        "1": {
```

```
"in": {
    "set": {
        "inv": false,
        "trim": 0,
        "bal": 0
    }
},
"col": 1,
"name": "",
"icon": 0,
"led": true,
"busmono": false,
"mute": false,
"fdr": -144,
"pan": 0,
"wid": 100,
"mon": "A",
"eq": {
    "on": false,
    "mdl": "STD",
    "mix": 100,
    "lg": 0,
    "lf": 60.13884,
    "lq": 0.99797,
    "leq": "SHV",
    "1g": 0,
    "1f": 129.8763,
    "1q": 0.99797,
    "2g": 0,
    "2f": 299.2472,
    "2q": 0.99797,
    "3g": 0,
    "3f": 699.4875,
    "3q": 0.99797,
    "4g": 0,
    "4f": 1499.788,
    "4q": 0.99797,
    "5g": 0,
    "5f": 2992.471,
    "5q": 0.99797,
    "6g": 0,
    "6f": 6013.884,
    "6q": 0.99797,
    "hg": 0,
    "hf": 11994.42,
    "hq": 0.99797,
    "heq": "SHV",
    "tilt": 0
},
"dyn": {
    "on": false,
    "mdl": "COMP",
    "mix": 100,
    "gain": 0,
    "thr": -10,
    "ratio": 3,
    "knee": 3,
    "det": "RMS",
    "att": 50,
    "hld": 20,
    "rel": 152.5652,
    "env": "LOG",
    "auto": true
},
"dynxo": {
    "depth": 6,
    "type": "OFF",
    "f": 1002.374
},
"dynsc": {
    "type": "OFF",
    "f": 1002.374,
    "q": 1.995882,
    "src": "SELF",
    "tap": "BUS"
},
"preins": {
```

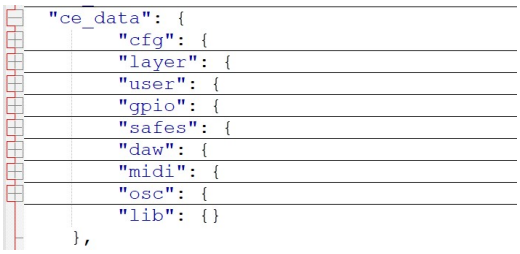
```
            "on": false,
            "ins": "NONE"
        },
        "send": {
            "MX1": {
                "on": false,
                "lvl": -144,
                "pre": false
            },
            "MX2": {},..."MX8": {}
        },
        "postins": {
            "on": false,
            "ins": "NONE"
        },
        "dly": {
            "on": false,
            "m": 0.1
        },
        "tags": ""
    },
    "2": {},..."4": {}
},
"mtx": {
    "1": {
        "in": {
            "set": {
                "inv": false,
                "trim": 0,
                "bal": 0
            }
        },
        "dir": {
            "1": {
                "on": false,
                "lvl": -144,
                "inv": false,
                "in": "OFF",
                "tap": "PRE"
            },
            "2": {}
        },
        "col": 1,
        "name": "",
        "icon": 0,
        "led": true,
        "busmono": false,
        "mute": false,
        "fdr": -144,
        "pan": 0,
        "wid": 100,
        "mon": "A",
        "eq": {
            "on": false,
            "mdl": "STD",
            "mix": 100,
            "lg": 0,
            "lf": 60.13884,
            "lq": 0.99797,
            "leq": "SHV",
            "1g": 0,
            "1f": 129.8763,
            "1q": 0.99797,
            "2g": 0,
            "2f": 299.2472,
            "2q": 0.99797,
            "3g": 0,
            "3f": 699.4875,
            "3q": 0.99797,
            "4g": 0,
            "4f": 1499.788,
            "4q": 0.99797,
            "5g": 0,
            "5f": 2992.471,
            "5q": 0.99797,
            "6g": 0,
            "6f": 6013.884,
```

```
            "6q": 0.99797,
            "hg": 0,
            "hf": 11994.42,
            "hq": 0.99797,
            "heq": "SHV",
            "tilt": 0
        },
        "dyn": {
            "on": false,
            "mdl": "COMP",
            "mix": 100,
            "gain": 0,
            "thr": -10,
            "ratio": 3,
            "knee": 3,
            "det": "RMS",
            "att": 50,
            "hld": 20,
            "rel": 152.5652,
            "env": "LOG",
            "auto": true
        },
        "dynxo": {
            "depth": 6,
            "type": "OFF",
            "f": 1002.374
        },
        "dynsc": {
            "type": "OFF",
            "f": 1002.374,
            "q": 1.995882,
            "src": "SELF",
            "tap": "BUS"
        },
        "preins": {
            "on": false,
            "ins": "NONE"
        },
        "postins": {
            "on": false,
            "ins": "NONE"
        },
        "dly": {
            "on": false,
            "m": 0.1
        },
        "tags": ""
    },
    "2": {},..."8": {}
},
"dca": {
    "1": {
        "name": "",
        "col": 1,
        "icon": 0,
        "led": false,
        "mute": false,
        "fdr": -144,
        "mon": "A"
    },
    "2": {},..."16": {}
},
"mgrp": {
    "1": {
        "name": "MGRP.1",
        "mute": false
    },
    "2": {},..."8": {}
},
"fx": {
    "1": {
        "mdl": "NONE",
        "fxmix": 100
    },
    "2": {},..."16": {}
},
"cards": {
```

```
        "wlive": {
            "autoin": "OFF",
            "meters": true,
            "1": {
                "cfg": {
                    "rectracks": "32",
                    "playmode": "PLAY"
                }
            },
            "2": {}
        }
    },
    "play": {
        "playall": true,
        "repeat": false
    },
    "rec": {
        "path": "WINGREC",
        "resolution": "24",
        "channels": "2"
    }
},
```

ce_data

ce_data contains all JSON structure elements representing the "Control Engine" settings for WING. The ce_data class contains the objects: ***cfg, layer, user, gpio, safes, daw, midi, osc, lib,*** as shown below:



Note that for ease of access and ***programming*** *using the native interface or OSC remote protocol, the ce_data* JSON *tree structure is appended to the ae_data tree structure.*

```
"ce_data": {
        "cfg": {
            "lights": {
                "btns": 25,
                "leds": 90,
                "meters": 40,
                "rgbleds": 25,
                "chlcds": 60,
                "chlcdctr": 50,
                "chedit": 80,
                "main": 80,
                "glow": 0,
                "patch": 0,
                "lamp": 0
            },
            "rta": {
                "homedisp": "1/3",
                "homecol": "BL50",
                "hometap": "IN",
                "eqdisp": "1/4",
                "eqcol": "BL75",
                "cheqtap": "PRE",
                "chflttap": "PRE",
                "eqdecay": "MED",
                "eqdet": "PEAK",
                "eqrange": 30,
                "eqgain": 0,
                "eqauto": true
            },
            "mtrsfc": {
                "in": "PRE",
                "bus": "POST",
                "main": "POST",
                "mtx": "POST",
                "dca": "PRE"
            },
            "mtrpage": {
                "in": "PRE",
                "bus": "POST",
                "main": "POST",
                "mtx": "POST",
                "dca": "PRE"
            },
            "mainmtr": "MAIN.1",
            "mainpos": "AUTO",
            "soloexcl": true,
            "selfsolo": true,
            "solofsel": false,
            "sof2solo": false,
            "layerlinkl": false,
            "layerlinkr": false,
            "autoview": false,
            "csctouch": true,
```

```
        "autosel_L": false,
        "autosel_C": false,
        "autosel_R": false,
        "fdrsel": false,
        "fdrres": "AUTO",
        "fdrspd": "MED",
        "fdrbanking": true,
        "soffdr": "L/C",
        "sofbutton": "AUTO",
        "sofframe": true,
        "sofmode": true,
        "seldblclick": true,
        "mousetchdis": false,
        "mousespd": 1.77,
        "usrmode": "CC",
        "tapflash": "ON",
        "srcdisp": true,
        "lockmtr": false,
        "timefmt": "24H",
        "datefmt": "YMD",
        "filesort": "A->Z"
    },
    "layer": {
        "L": {
            "sel": 5,
            "1": {
                "ofs": 0,
                "name": "CH1-12",
                "1": {
                    "type": "CH",
                    "i": 1,
                    "dst": 1
                },
                "2": {},..."24": {}
            },
            "2": {}..."7": {}
            }
        },
        "C": {
            "sel": 2,
            "1": {
                "ofs": 0,
                "name": "DCA",
                "1": {
                    "type": "DCA",
                    "i": 1,
                    "dst": 1
                },
                "2": {},..."16": {}
            },
            "2": {}..."6": {}
        },
        "R": {
            "sel": 1,
            "1": {
                "ofs": 0,
                "name": "MAIN",
                "1": {
                    "type": "BUS",
                    "i": 17,
                    "dst": 1
                },
                "2": {},..."16": {}
            },
            "2": {}..."7": {}
        }
    },
    "user": {
        "sel": 1,
        "mode": "USER",
        "cmode": "HA",
        "gpio": {
            "1": {
                "bu": {
                    "mode": "OFF",
                    "name": "GPIO 1"
                }
```

```
            },
            "2": {},..."4": {}
        },
        "user": {
            "1": {
                "bu": {
                    "mode": "OFF",
                    "name": ""
                },
                "bd": {
                    "mode": "OFF",
                    "name": ""
                }
            },
            "2": {},..."4": {}
        },
        "daw1": {
            "1": {
                "bu": {
                    "mode": "DAWBTN",
                    "name": "STOP",
                    "btn": "T1"
                },
                "bd": {
                    "mode": "DAWBTN",
                    "name": "REWIND",
                    "btn": "T4"
                }
            },
            "2": {},..."4": {}
        },
        "daw2": {},..."daw4": {}
        },
        "1": {
            "1": {
                "led": false,
                "col": 1,
                "enc": {
                    "mode": "OFF",
                    "name": ""
                },
                "bu": {
                    "mode": "OFF",
                    "name": ""
                },
                "bd": {
                    "mode": "OFF",
                    "name": ""
                }
            },
            "2": {},..."4": {}
        },
        "2": {},..."16": {},
        "cuser": {
            "1": 1,
            "2": 1,
            "3": 1
        }
    },
    "gpio": {
        "1": {
            "mode": "TGLNO",
            "gpstate": false
        },
        "2": {},..."4": {}
    },
    "safes": {
        "ch": "",
        "aux": "",
        "bus": "",
        "main": "",
        "mtx": "",
        "dca": "",
        "mute": "",
        "fx": "",
        "source": {
            "LCL": "",
```

```
            "AUX": "",
            "A": "",
            "B": "",
            "C": "",
            "SC": "",
            "USB": "",
            "CRD": "",
            "MOD": "",
            "PLAY": "",
            "AES": "",
            "USR": "",
            "OSC": ""
        },
        "output": {
            "LCL": "",
            "AUX": "",
            "A": "",
            "B": "",
            "C": "",
            "SC": "",
            "USB": "",
            "CRD": "",
            "MOD": "",
            "REC": "",
            "AES": ""
        },
        "area": {
            "L": "0",
            "C": "0",
            "R": "0"
        },
        "custom": "",
        "setup": ""
    },
    "daw": {
        "on": true,
        "conn": "USB",
        "emul": "MCU",
        "config": "CC",
        "ccup": false,
        "disjog": false,
        "preset": "-"
    },
    "midi": {
        "enchctl": "OFF",
        "enfxctl": "OFF",
        "encustctl": "OFF",
        "ensysex": "OFF",
        "enmidicc": "OFF"
    },
    "osc": {
        "ronly": false
    },
    "lib": {}
},
```


More JSON files

WING desk provides more `JSON` files. Indeed, `JSON` format is also used to save/store channel, library, and effect presets. These files are created as you save presets and libraries that help you setup your system faster down the road.