WING
PERSONAL MIXING CONSOLE

OSC Documentation

for WING

Table of Contents

Introduction

My name is Patrick-Gilles Maillot and I am authorized by Behringer to publish and maintain the “OSC Remote Control Documentation for WING”, yet I am not a MusicTribe employee.

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 general public between 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 a number of 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.

General features of the WING console

The Behringer WING provides 48-channel, 28-bus mixing with 24 motorized, touch-sensitive 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.

WING focuses on sound sources as the reason for any mixing, having properties like headamp gain, phantom power, source mutes and metering. Sources can be personalized with color, icon, name, and several tags for grouping and filtering purposes. The 48 inputs and 28-channel bus mixes can all be in mono/stereo or mid-side modes, keep headamp parameters like gain and phantom power, and with specific source mutes and metering and provide dynamics, EQ and FX processing. They can also be personalized with their own color, icon, name, and several 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, and 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 of 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 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 374 inputs 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.

Sources vs. Inputs

Unlike many digital or analogue desks, WING makes a clear separation between Sources and Input channels; Normally, 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[1] mode, own headamp parameters like gain and phantom power, with specific source mute and metering. They can also be given a color, icon, name and several 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.

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[2], 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.

[1] Mid/Side processing is a highly effective way of making adjustments to the spatialization perception of a mix or master. The Mid channel is the center of a stereo image. When the Mid channel is boosted, the listener perceives a more centered (mono) sound to the audio. The Side channel is the edges of a stereo image. When the Side channel is boosted, the listener perceives a more spacious (wider) sound to the audio.

[2] Described in a separate document. Refer to https://github.com/pmaillot/wapi

WING Internal Data

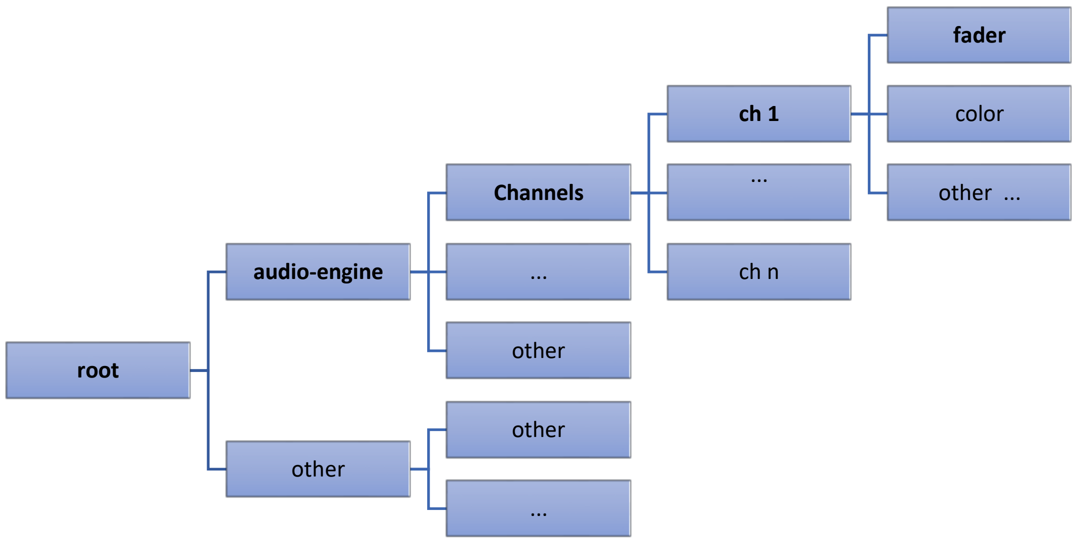
Like all digital or programmable devices, WING relies on an internal set of parameters that are stored/saved in non-volatile memory. This enables you to find the console in the same state you left it when powering it OFF.

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.

The WING tree is more than 25000 elements! In order 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:

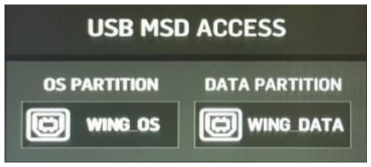


Computers use specific data structures to represent trees. WING uses one of them, based on JSON[3] notation. It is important to know/understand the list of sub-trees (nodes), and leaves (parameters) WING contains as this is how you can access to data. More detail on the WING data set is provided in appendix.

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

WING File System

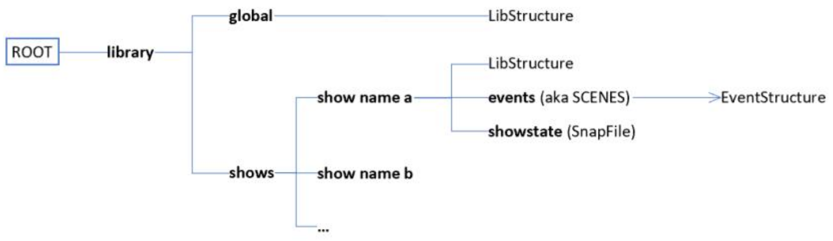
At the difference of the X32, WING can be directly connected to a computer via USB; There are two ways WING can be visible to your computer, depending on the setting of the `SETUP→GENERAL` screen (shown below):

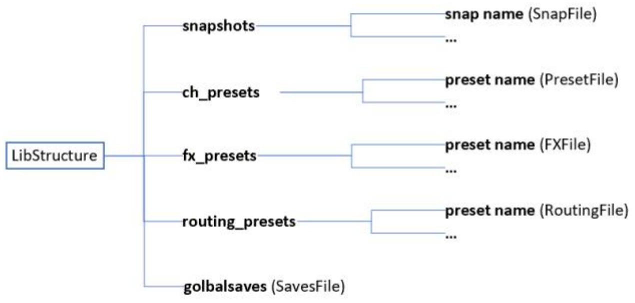


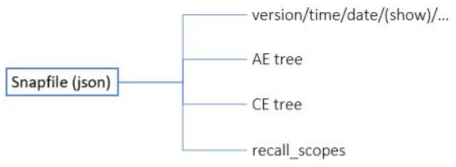
WING can be seen as an `OS PARTITION`, or a directory where you can deposit the FW release you will use to boot from at next power up or reboot. Use with caution!

If the choice in `SETUP→GENERAL` is set to `DATA PARTITION`, the 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.***

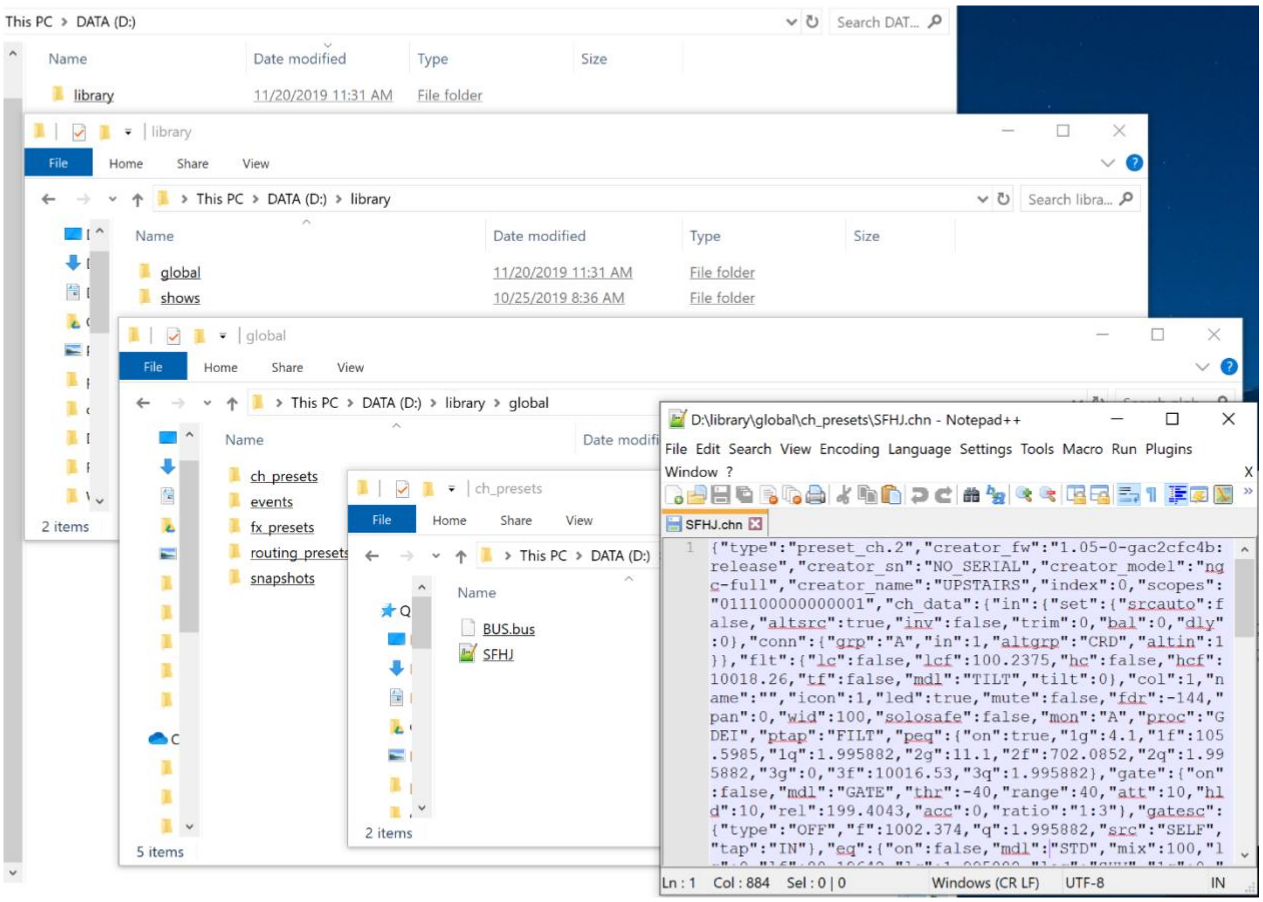
When connected, the WING file system is as follows (nodes in bold are real folder names):







Below is a screenshot of the consecutive opening of directories `library→globals→chi_presets`, and opening file `SFHJ.chn` (a JSON structure file), the PC being in DATA ACCESS mode over a USB connection:



This PC > DATA (D:)

Name	Date modified	Type	Size
library	11/20/2019 11:31 AM	File folder	

This PC > DATA (D:) > library

Name	Date modified	Type	Size
global	11/20/2019 11:31 AM	File folder	
shows	10/25/2019 8:36 AM	File folder	

2 items

This PC > DATA (D:) > library > global

ch_presets
events
fx_presets
routing_presets
snapshots

5 items

ch_presets

BUS.bus
SFHJ

2 items

D:\library\global\ch_presets\SFHJ.chn - Notepad++

```
{"type":"preset_ch.2","creator_fw":"1.05-0-gac2cfc4b:
release","creator_sn":"NO_SERIAL","creator_model":"ng
c-full","creator_name":"UPSTAIRS","index":0,"scopes":
"011100000000001","ch_data":{"in":{"set":{"srcauto":f
alse,"altsrc":true,"inv":false,"trim":0,"bal":0,"dly"
:0},"conn":{"grp":"A","in":1,"altgrp":"CRD","altin":1
}},"flt":{"lc":false,"lcf":100.2375,"hc":false,"hcf":
10018.26,"tf":false,"mdl":"TILT","tilt":0},"col":1,"n
ame":"","icon":1,"led":true,"mute":false,"fdr":-144,"
pan":0,"wid":100,"solosafe":false,"mon":"A","proc":"G
DEI","ptap":"FILT","peq":{"on":true,"1g":4.1,"1f":105
.5985,"1q":1.995882,"2g":11.1,"2f":702.0852,"2q":1.99
5882,"3g":0,"3f":10016.53,"3q":1.995882},"gate":{"on"
:false,"mdl":"GATE","thr":-40,"range":40,"att":10,"hl
d":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,"l
```

Ln : 1 Col : 884 Sel : 0 | 0 Windows (CR LF) UTF-8 IN

Remote communication with WING

WING communicates via ports `2222 [native UDP, TCP] and 2223 [OSC, UDP]`;

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)
```

General OSC communications take place over communication port 2223

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”

At the time of this document, WING’s OSC remote protocol enables 1 (one) client subscribing to data (so called “unsolicited” messages). Subscriptions have to be kept alive; they automatically die after 10 seconds.

Access to WING Internal Data from remote programs

WING offers several remote protocols with the capability to access (read or write) parameters of its internal structures and take full advantage of the numerous features of the digital desk, including remote control. One of them is WING's native (binary) interface and is covered in a separate document. This document focuses on **OSC**.

WING hosts an **OSC** compliant remote protocol server that offers access to the full set of features of the desk.

OSC Remote Protocol

WING includes an OSC Remote Protocol server. This enable easy access to remote features for many professional, sound applications and extensions offered by third parties.

OSC remote control enables reading and modifying (if possible) all parameters included in the `ae_data` and `ce_data` JSON structures; In order to allow this, `ce_data` parameters are included under the `$ctl` subtree in the main parameter tree.

WING OSC server implementation complies with the OSC standard[4] 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, connected clients will be replied onto their calling port. 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 bits a multiple of 32),

`blob` (An int32 size count, followed by that many 8-bit bytes of arbitrary binary data, followed by 0-3 additional zero bytes to make the total number of bits a multiple of 32).

[4] See http://opensoundcontrol.org/spec-1_0

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 (the use of '`?`' and '`*`' in Address Patterns) are not allowed.

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.

WING OSC Messages

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

In the text shown below, 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 Byte exchanges are displayed below (OSC is a binary protocol)

```
->W,    4 B: 2f3f0000
W->,   80 B:
2f3f00002c73000057494e472c3139322e3136382e312e37312c50474d2c6e67632d66756c6c2c4e4f5f53455249414
c2c312e30372e322d34302d6731623162323932623a646576656c6f7000000000
```

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 mains ways to gain access to WING data: using one-parameter-at-a-time or using “nodes”.

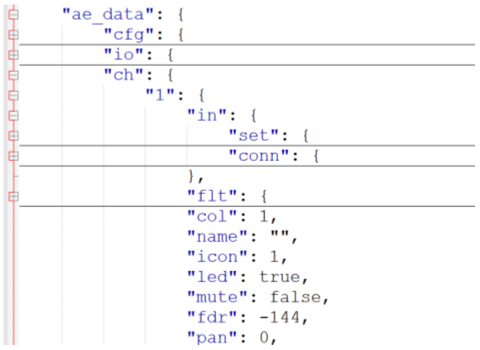
WING “nodes” are a great way to access multiple parameters at a time, and therefore maximize communication bandwidth with the console. Nodes are represented as **string** OSC Data Type and are zero terminated (`\0` byte ending the string).

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:
/~~~,ssssssssssssssss~~~$stat~~~cfg~$syscfg~io~~ch~~aux~bus~main~~~~mtx~dca~mgrp~~~~fx~~cards~~
~play~~~~rec~$ctl~~~~
```

Retrieving a WING single parameter is quite easy: You have to 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 the case for the fader value of a channel strip for example, or its mute state. Channel Strip 1 fader is represented as follows:



```
    "ae_data": {
        "cfg": {
        "io": {
        "ch": {
            "1": {
                "in": {
                    "set": {
                    "conn": {
                },
                "flt": {
                "col": 1,
                "name": "",
                "icon": 1,
                "led": true,
                "mute": false,
                "fdr": -144,
                "pan": 0,
```

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 interpretations of two 32bits big-endian float data values, each represented 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: 2f63682f312f6664720000002c7366660000000002d6f6f0000000000c3100000
```

Depending on the OSC Address Pattern, WING returns '`,s`' for strings or enums, '`,sff`' (ascii, raw pos, float value) for floats, '`,sfi`' (ascii, raw pos, 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: 2f63682f312f6d75746500002c73666900000000310000003f80000000000001
```

It should be noted that WING will accept both OSC path or hash data as 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.

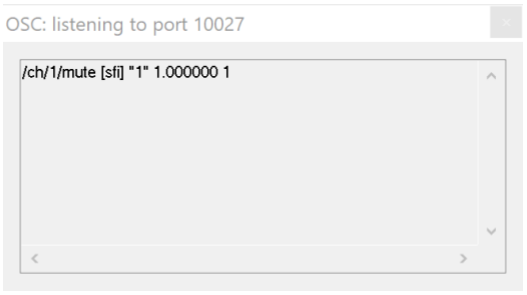
Receiving OSC data on a specific port

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. In order 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.



OSC: listening to port 10027

```
/ch/1/mute [sfi] "1" 1.000000 1
```

Writing (Set) Parameter and Node data

Single Parameters

OSC can also be used to set or modify WING data. Taking the fader and mute examples above, we can modify their respective data 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.

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]
```

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~
```

if an error occurred during the execution of the command.

```
->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]
```

OSC: Special Cases

Dynamic JSON Structure changes

As parameters get changed on the WING console, its JSON structure tree evolves to reflect the change; 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 the dynamic allocation of effect or plugins, generating 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 be able to get and possibly set effect parameter values.

The WING effects and plugins, and their respective parameters are listed later in this document[5].

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~,ssssssssssssssss~~~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~~~,ssssssssssssssssss~mdl~fxmix~~~$esrc~~~$emode~~$a_chn~~$a_pos~~mix~g~~~31~~63~~125~250
~500~1k~~2k~~4k~~8k~~16k~
```

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

```
->W,   12 B: /fx/1/125~~~
```

[5] Please refer to the "Effects" paragraph

```
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]
```

OSC Tag Type 'blob' use

WING OSC server implementation supports the 'blob' OSC Tag type, enabling the use of 'native' commands[6] 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 blob.

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

Blob types typically apply on WING nodes in order 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 blob data [all bytes sent shown as hex data]

```
/ ,b dddde
```

Data actually sent (in hex): `->W,   16 B: 2f0000002c62000000000002dddde0000`

WING's reply is:

```
W->,  376 B: /~~~,b~~361 bytes:
df001497a0043900000524737461740553544154450000df000dedca7af90000036366670000000df0011f89818a6000
007247379736366670000000df000f294f7794000002696f03492f4f0000df001370b101390000026368074348414e4e
454c0000df00188fa3078d0000036175780b4155582043484e4e454c0000df0010f46c185e00000362757303425555
30000df001204d3a3a80000046d61696e044d41494e0000df0013f82a5af20000036d7478064d41545249580000df00
10e313aeff00000364636103444341000df0018d252398b0000046d6772700a4d5554452047524f55500000df00134
73c91340000026678074546464543545300000df001eb4296fc900000563617264730f455850414e53494f4e20434152
```

[6] Detailed information on native commands is provided in a separate document

```
44530000df001457297a28000004706c617906504c415945520000df0015fab1762c000003726563085245434f52444
5520000df0015cbb95143000004246374 6c07434f4e54524f4c0000de~~~
```

Lots of information are returned either as string, or more often as blob. 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 ddde`, or

`/ ,b d7df17c242ddde`

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

`->W,   24 B: 2f2463746c2f6461770000002c62000000000002ddde0000` or

`->W,   20 B: 2f0000002c62000000000007d7df17c242ddde00`

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

```
W->,  764 B: /$ctl/daw~~~,b~~744 bytes:
df001e3cb129d50000026f6e0a44415720454e41424c4500400000000000000000001df00244e5c7f34000004636f6e6e0
a434f4e4e454354494f4e00500000203444494e000355534200df0023e5681680000004656d756c09454d554c4154494f
4e00500002034d43550003485549 00df006d42701ca9000006636f6e666967000050000402434314435553544f4d204
34f4e54524f4c53204f4e4c59044d5354520a53494e474c45204d4355084d535452314558540e4d4355202b20455854
454e444552084d53545232455854114d4355202b20327820455854454e444552df002aae1538a400000463637570145
55345205550504552204343204 64f5220444157004000000000000000000001df0093892e512d00000670726573657412 4c
415354204c4f41444544205052455345540050000 8012d012d06637562617365064355424153 45046c697665044c495
645066c6f67696378074c4f4749432058066e75656e646f064e55454e444f0870726f746f6c730950524f20544f4f
4c5306726561706572065245415045520973747564696f6f6e650a53545544494f204f4e45df001bbeefaeab0000032
46f6e06444157204f4e0040000000000000000001df00239631559f000006246270616765 0b425554544f4e2050414745
004000000000000000004df002d012dc5460000092462746e746f7563681242544e53454c20464144455220544f55434
80040000000000000000001df0026775c19c20000082462746e76706f740c42544e53454c20562d504f54004000000000
00000001df002942aeb92800000a2462746e7265637264790d42544e53454c20524543524459004000000000000000000
1df0025fccfbe070000082462746e6175746f0b42544e53454c204155544f0040000000000000000001df002685cdce3f
0000082462746e7673656c0c42544e53454c20562d53454c0040000000000000000001df002915abd96800000a2462746
e696e736572740d42544e53454c20494e53455254004000000000000000000001de
```

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~~~,ssssssssssssss~on~~conn~~~~emul~~~~config~~ccup~~~~preset~~$on~$bpage~~$btntouch~~
~$btnvpot~~~~$btnrecrdy~~$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).

We can also use the blob Type Tag 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:

```
->W,   24 B: /~~~,b~~12 bytes: d7beefaeabd400000001ddde

->W,   28 B: /$ctl/daw/$on~~~,b~~3 bytes: 01ddde~
```

In the binary data sent with the line above, the segment `d400000001` 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:

```
->W,   24 B: /~~~,b~~12 bytes: d7beefaeabd400000000ddde

->W,   28 B: /$ctl/daw/$on~~~,b~~3 bytes: 00ddde~
```

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

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

```
/$ctl/daw/$on ,b dc
->W,   28 B: 2f2463746c2f6461772f246f6e0000002c62000000000001dc000000
```

WING returns the following which includes the hash value for `/$ctl/daw/$on` and the current value (WING native coding) for the parameter: `00`

```
/$ctl/daw/$on~~~,b~~7 bytes: d7beefaeab00de ~
W->,   32 B: 2f2463746c2f6461772f246f6e0000002c62000000000007d7beefaeab00de00
```

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

Subscribing to OSC Data

There are two main types of subscription: binary or OSC messages.

At the time of this document, subscriptions are valid for all OSC WING messages only, and a maximum of 1 subscription can be active at any time, provided to the last requestor.

Subscriptions must be renewed every `10 seconds` in order to keep alive.

`/*b~` will enable receiving unsolicited binary messages

Binary messages are formatted exactly as the binary/native interface and therefore can be sent back to the console with no change.

`/*s~` will request OSC messages

OSC messages are received as triplets of data, as presented above[7]; 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).

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 such as shown below:

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

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

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
```

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
```

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 and slots 9-16 dedicated to premium effects and standard effects, respectively. As one can expect, premium effect slots can be running standard effects too.

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, 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, VSS3 REVERB,
```

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, 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
```

Effects can be used as dedicated inserts at two defined location within the audio path: pre and post xxx.

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[8] elements in the actual WING structure of a non-affected effect slot, resulting in the following JSON structure:

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

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

```
                    “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 particular effect slot, thus enabling for example up to 16 different copies[9] of the same effect to be active on every effect slots, with differentiated parameters for each slot.

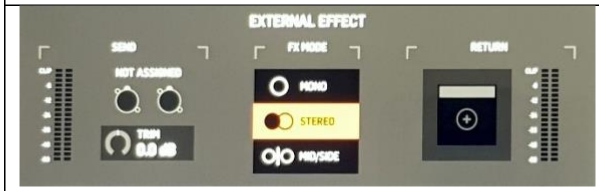
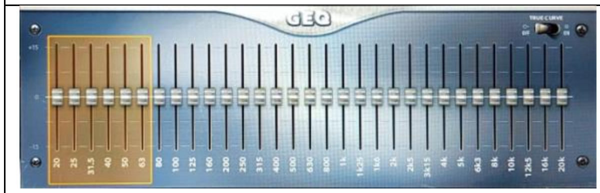
The tables below will list the effect names and parameters, and the parameter types associated with each known effect.

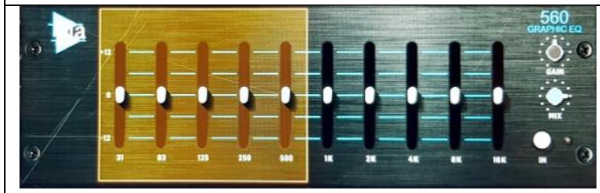
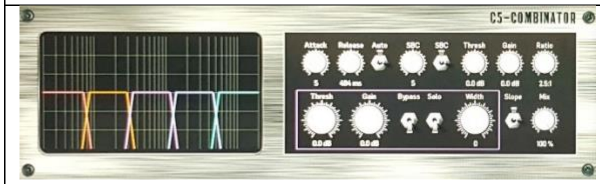
[9] For standard effects, 8 for premium effects

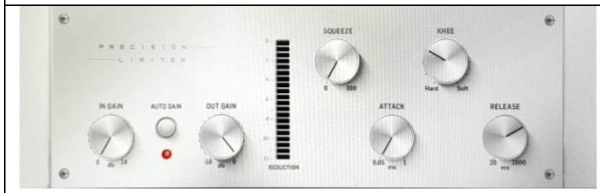
Effects and Plugins’ Parameters list

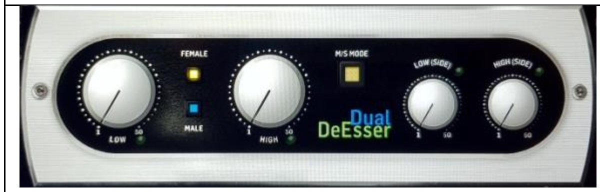
In the (long) tables below, we list all known/exposed effects and plugins available with the WING digital console, along with their name, type, and min/max/step/list values; We therefore present Standard Effects, Premium effects, Filter Plugins, Gate Plugins, EQ Plugins, and Compressor Plugins.

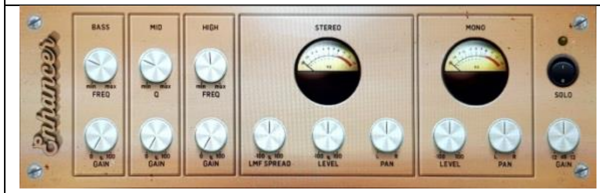
Standard effects

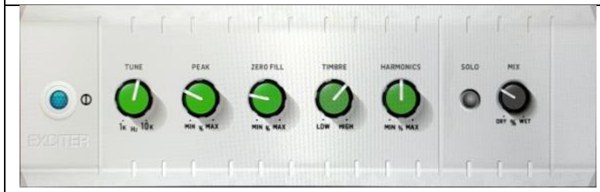
	None		
	0	***“mdl”:***	*NONE*
 EXTERNAL EFFECT SEND NOT ASSIGNED TRIM 0.0 dB FX MODE MONO STEREO MID/SIDE RETURN CLIP	External		
	0	***“mdl”:***	*EXT*
	1	***“egrp”:***	*str [OFF, LCL, AUX, A, B, C, SC, USB, CRD, MOD, PLAY, AES] ext grp*
	2	***“ein”:***	*int [1…64] ext in*
	3	***“emode”:***	*str [M, ST, 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*

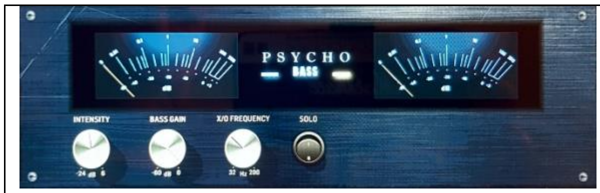
	28 **“8k”**: linf [-15, 15, 121] dB 29 **“10k”**: linf [-15, 15, 121] dB 30 **“12k5”**: linf [-15, 15, 121] dB 31 **“16k”**: linf [-15, 15, 121] dB 32 **“20k”**: linf [-15, 15, 121] dB
 560 GRAPHIC EQ GAIN MIX IN 31 63 125 250 500 1K 2K 4K 8K 16K	PIA 560 GEQ 0 **“mdl”**: PIA 1 **“mix”**: linf [0, 125, 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
 C5-COMBINATOR Attack Release Auto SBC SBC Thresh Gain Ratio Thresh Gain Bypass Solo Width Slope Mix	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`

	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`

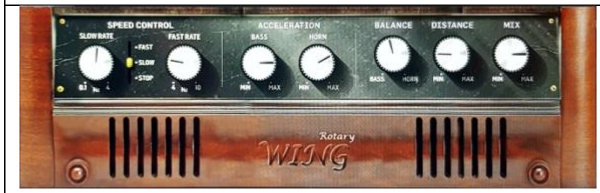
	Ultra Enhancer
	
	`0 "mdl":   ENHANCE`
	`1 "stlv":   linf [-100, 100, 201] %, st lvl`
	`2 "lmf":    linf [-100, 100, 201] %, lmf spread`
	`3 "lmvl":   linf [-100, 100, 201] %, mono lvl`
	`4 "st":     linf [-100, 100, 201] %, st pan`
	`5 "m":      linf [-100, 100, 201] %, mono pan`
	`6 "bass":   linf [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 "solo":   int [0, 1] solo`



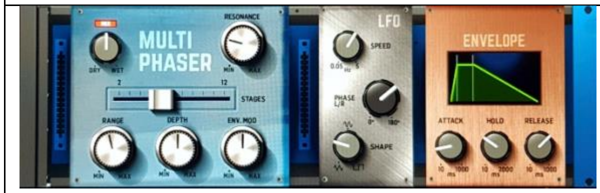
Psycho Bass

```
 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
```



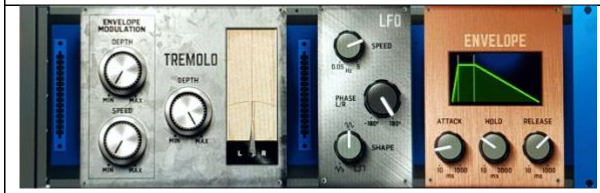
Rotary Speaker

```
 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
```



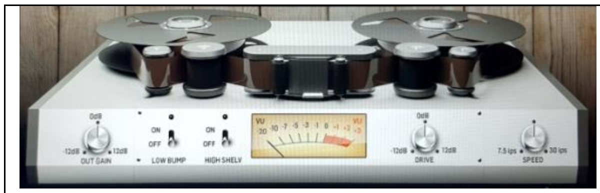
Phaser

```
 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
```



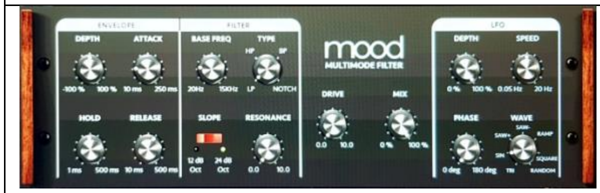
Tremolo Panner

```
 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
```

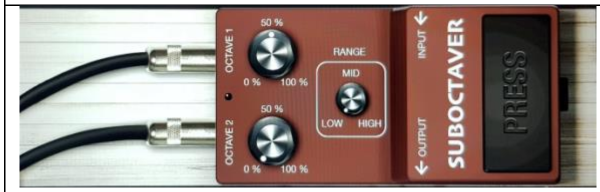


Mood Filter

```
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
```

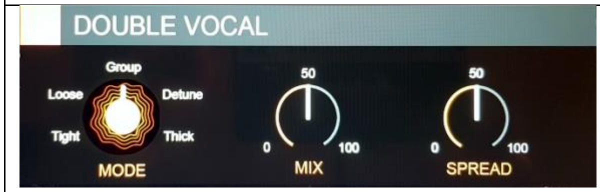
Bodyrez

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



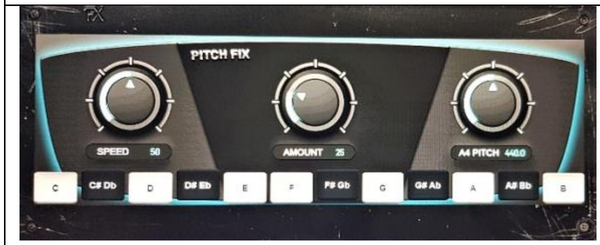
Sub Octaver

```
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
```



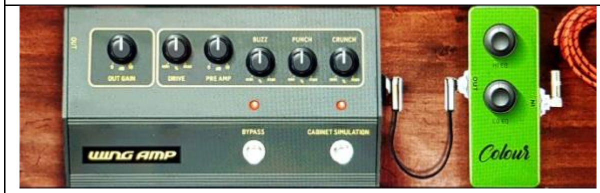
Double Vocal

```
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
```



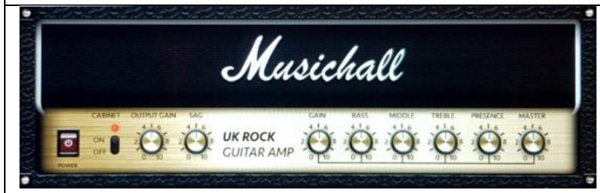
Pitch Fix

```
 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]
```



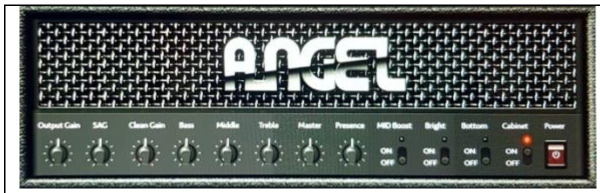
Rack Amp

```
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
```



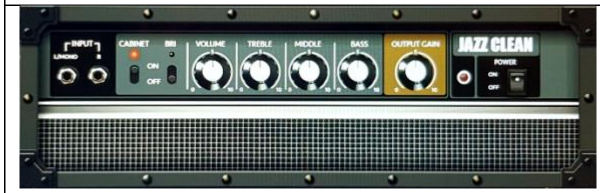
UK Rock Amp

```
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
```



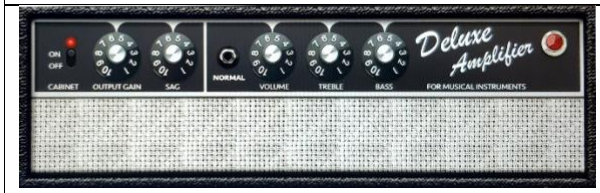
Angel Amp

```
 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
```



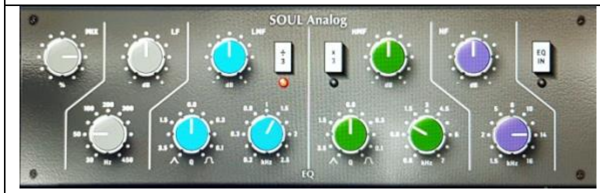
Jazz Clean Amp

```
 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
```



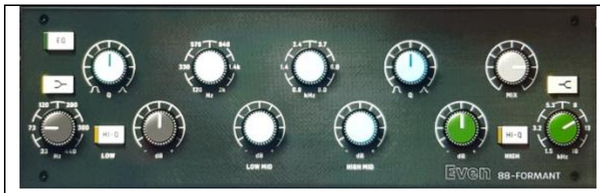
Deluxe Amp

```
 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
```



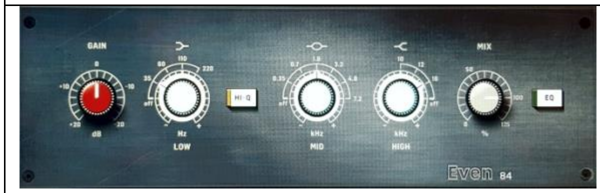
Soul Analogue

```
 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
```



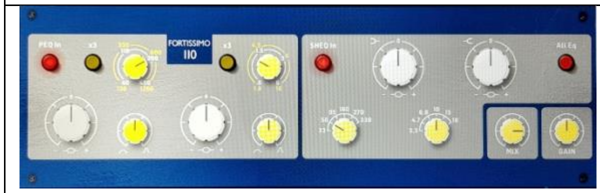
Even 88 Formant

```
 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
```



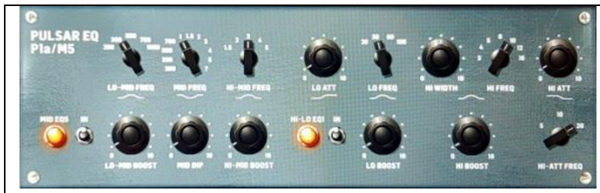
Even 84

```
 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
```



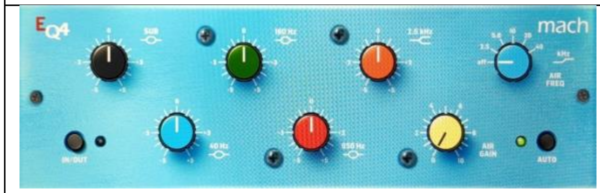
Fortissimo110

```
 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
 4 “lmq”:   linf [0, 10, 101] lm q
 5 “lmf3”:  int [0, 1] lm /3
 6 “hmf”:   linf [0, 10, 101] hm freq
 7 “hmg”:   linf [-5, 5, 101] hm gain
 8 “hmq”:   linf [0, 10, 101] hm q
 9 “hmf3”:  int [0, 1] hm x3
10 “shv”:   inf [0, 1] shv on
11 “lf”:    str [33, 56, 95, 160,
                 270, 460] lf freq
11 “lg”:    linf [-5, 5, 101] lf gain
13 “hf”:    str [3k3, 4k7,6k8, 10k,
                 15k, 18k] hf freq
14 “hg”:    linf [-5, 5, 101] hf q
15 “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
 4 “1lf”:    str [20, 30,60, 100] Hz, lf freq
 5 “1hw”:    linf [0, 10, 101] hf wid
 6 “1hb”:    linf [0, 10, 101] hf boost
 7 “1hf”:    str [3k, 4k, 5k, 8k, 10k,
                  12k, 16k] Hz, hf freq
 8 “1hatt”: linf [0, 10, 101] hf att
 9 “1hattf”:str [5k, 10k, 20k] hf att
10 “eq5”:    inf [0, 1] eq5 on
11 “5lb”:    linf [0, 10, 101] lm boost
12 “5lf”:    str [200, 300, 500, 700,
                  1k] Hz, lf freq
13 “5md”:    linf [0, 10, 101] mid dip
14 “5mf”:    str [200, 300, 500, 700, 1k, 1k5,
                  2k, 3k, 4k, 5k,7k] Hz, mid freq
15 “5hb”:    linf [0, 10, 101] HM boost
16 “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

PRE DLY 50 MS · SIZE 50 · DECAY 1.57 S · BASS MULT 0.95 · DAMPING 5K7 HZ · DIFFUSION 25 · HALL LAYER 1

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
```

PRE DLY 40 MS · SIZE 50.0 M · DECAY 2.51 S · BASS MULT 1.21 · DAMPING 5K7 HZ · DIFFUSION 40 · ROOM LAYER 1

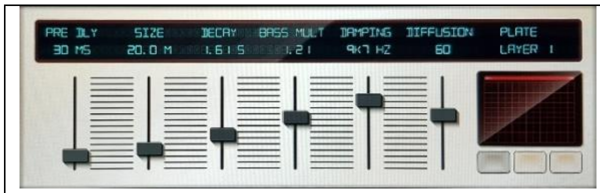
Room Reverb

```
 0 “mdl”:  ROOM
 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
```

PRE DLY 30 MS · SIZE 25.0 M · DECAY 1.35 S · BASS MULT 1.15 · DAMPING 5K0 HZ · DIFFUSION 50 · CHAMBER LAYER 1

Chamber Reverb

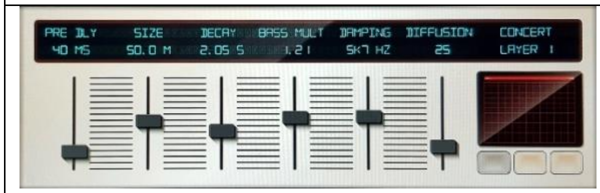
```
 0 “mdl”:  CHAMBER
 1 “pdel”: int [0…200] ms, pre-delay
 2 “size”: linf [4, 76, 145] m, room size
 3 “dcy”:  logf [.3, 25, 101] s, decay
 4 “mult”: logf [.25, 4, 101] bass multiplier
 5 “damp”: logf [1k, 20k, 51] Hz, damping
 6 “lc”:   logf [20, 400, 51] Hz, low cut
 7 “hc”:   logf [200, 20k, 51] 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
```



PRE DLY	SIZE	DECAY	BASS MULT	DAMPING	DIFFUSION	PLATE
30 MS	20.0 M	1.61 S	1.21	9K7 HZ	60	LAYER 1

Plate Reverb

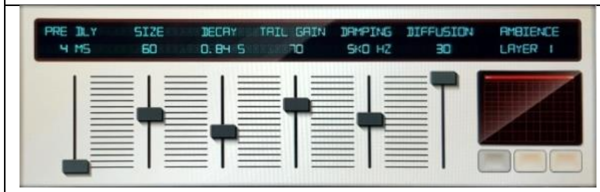
```
 0 “mdl”:  PLATE
 1 “pdel”: int [0…200] ms, pre-delay
 2 “size”: linf [4, 76, 145] m, room size
 3 “dcy”:  logf [.3, 25, 101] s, decay
 4 “mult”: logf [.25, 4, 101] bass multiplier
 5 “damp”: logf [1k, 20k, 51] Hz, damping
 6 “lc”:   logf [20, 400, 51] Hz, low cut
 7 “hc”:   logf [200, 20k, 51] Hz, 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
```



PRE DLY	SIZE	DECAY	BASS MULT	DAMPING	DIFFUSION	CONCERT
40 MS	50.0 M	2.05 S	1.21	5K7 HZ	25	LAYER 1

Concert Reverb

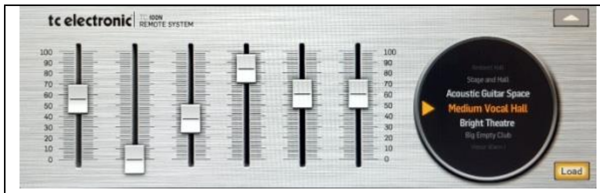
```
 0 “mdl”:  CONCERT
 1 “pdel”: int [0…200] ms, pre-delay
 2 “size”: linf [20, 76, 113] m, room size
 3 “dcy”:  logf [.3, 29, 51] s, decay
 4 “mult”: logf [.25, 4, 101] bass multiplier
 5 “damp”: logf [1k, 20k, 51] Hz, damping
 6 “lc”:   logf [20, 400, 51] Hz, low cut
 7 “hc”:   logf [200, 20k, 51] 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
```



PRE DLY	SIZE	DECAY	TAIL GAIN	DAMPING	DIFFUSION	AMBIENCE
4 MS	60	0.84 S	70	5K0 HZ	30	LAYER 1

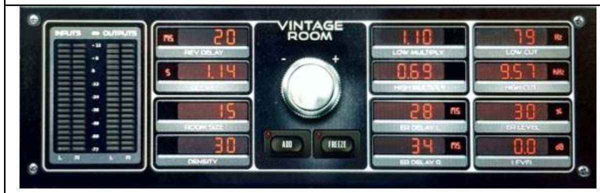
Ambiance

```
 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
```



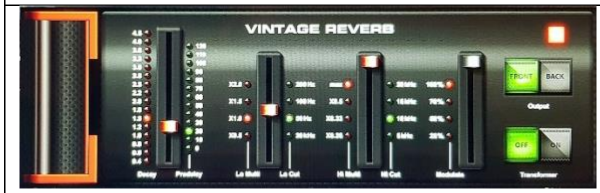
VSS3 Reverb

```
 0 “mdl”:   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
11 “erl”:   linf [0, 200, 201] ms, e. ref. left
12 “err”:   linf [0, 200, 201] ms, e. ref. right
13 “add”:   int [0, 1] add
```



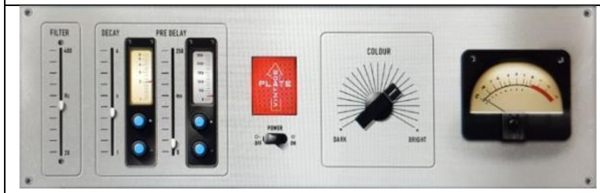
Vintage Room

```
 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
```



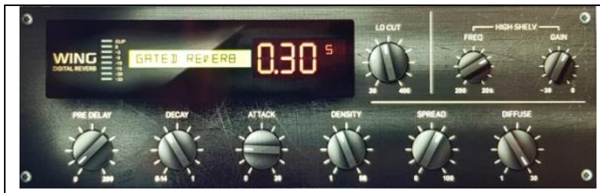
Vintage Reverb,

```
 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
```



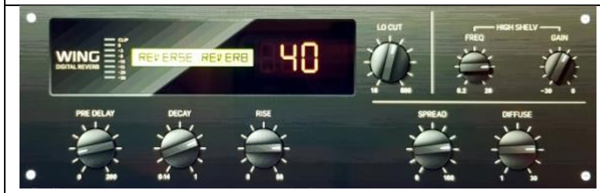
Vintage Plate

```
 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
```



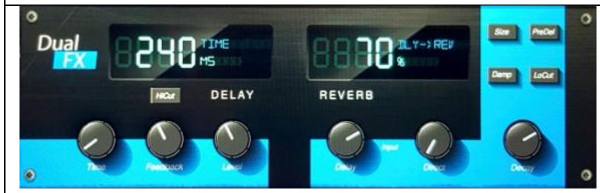
Gated Reverb

```
 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
```



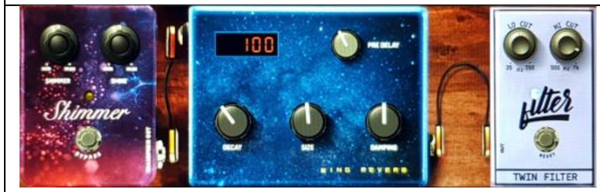
Reverse Reverb

```
 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
```



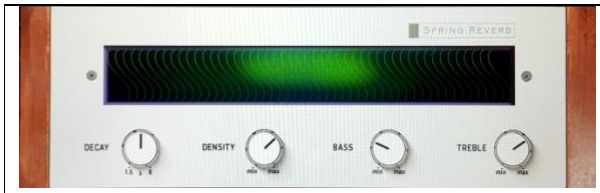
Delay/Reverb

```
 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
```



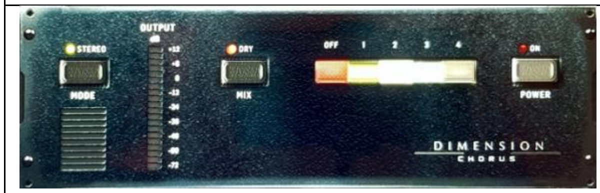
Shimmer Reverb

```
 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
```



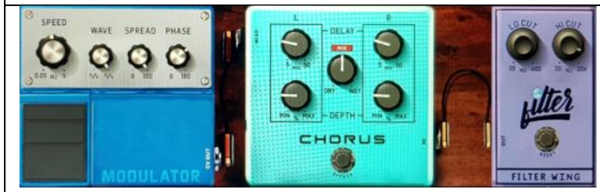
Spring Reverb

```
 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
```



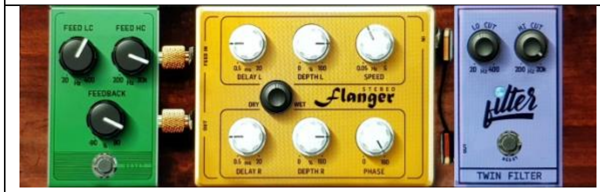
Dimension CRS

```
 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
```



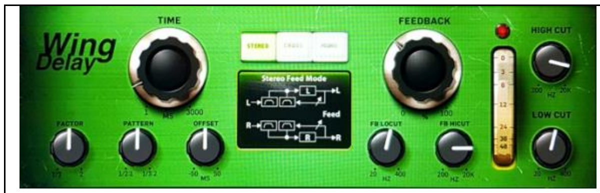
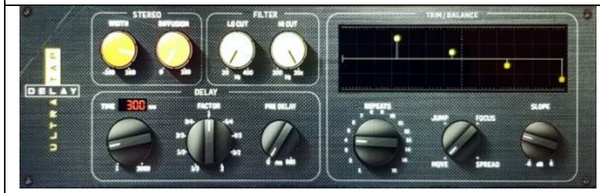
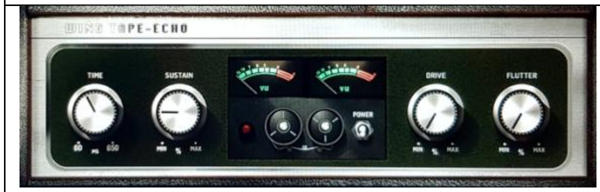
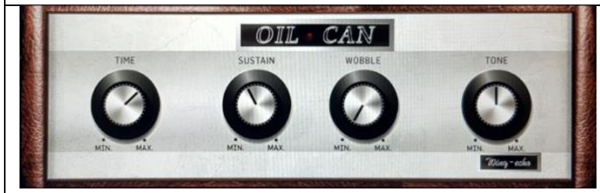
Stereo Chorus

```
 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
```



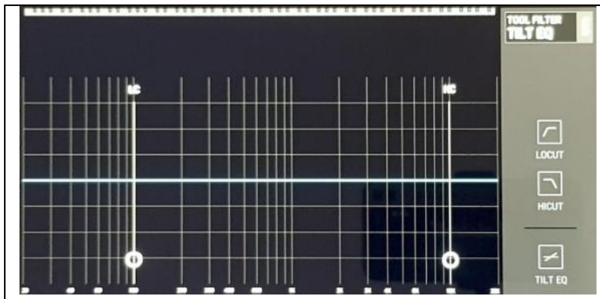
Stereo Flanger

```
 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
```

 Wing Delay TIME FEEDBACK HIGH CUT LOW CUT FACTOR PATTERN OFFSET FB LOCUT FB HICUT Stereo Feed Mode	Stereo Delay 0 *“mdl”:* *ST-DL* 1 *“time”:* *linf [1, 3000, 3000] ms, time* 2 *“mode”:* *str [ST, X, M] mode* 3 *“fact”:* *str [1/3, 1/2, 2/3, 3/4, 1, 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 STEREO FILTER TRIM / BALANCE DELAY REPEATS JUMP FOCUS SLOPE MOVE SPREAD	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*
 WING TAPE-ECHO TIME SUSTAIN DRIVE FLUTTER POWER	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*

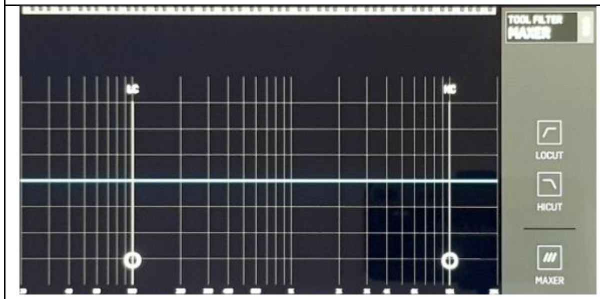
	BBD Delay *0 “mdl”: BBD-DL* *1 “dly”: linf [0, 100, 1001] time* *2 “feed”: linf [0, 100, 101] %, feed*
	Stereo Pitch *0 “mdl”: PITCH* *1 “semi”: int [-12…12] semitones* *2 “cent”: int [-50…50] cent* *3 “dly”: linf [0, 500, 501] ms, delay* *4 “lc”: logf [20, 400, 51] Hz, low cut* *5 “hc”: logf [200, 20000, 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*

Filter plugins



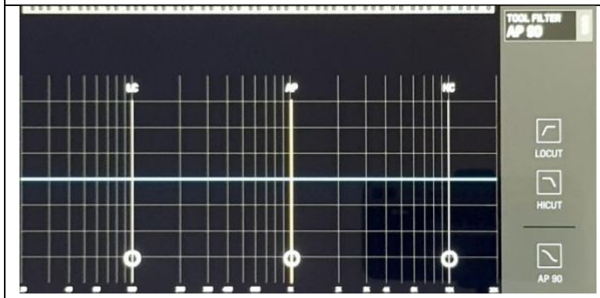
Tilt Filter

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



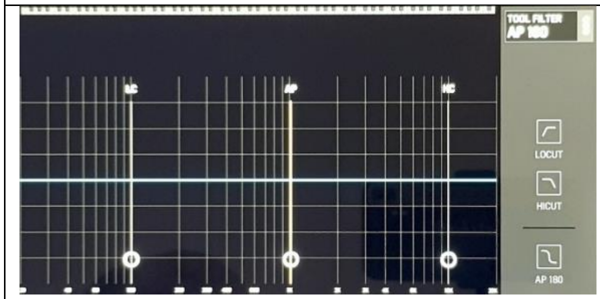
Maxer Filter

```
0 “mdl”:   TILT
1 “low”:   linf [0, 100, 101] %, low cont
2 “proc”:  linf [0, 100, 101] %, high proc
```



AP90axer Filter

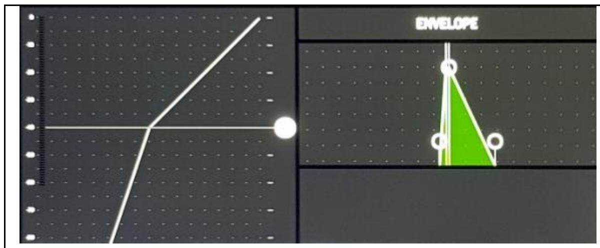
```
0 “mdl”:   AP1
1 “freq”:  logf [100, 10000, 100] Hz, freq
```



AP180 Filter

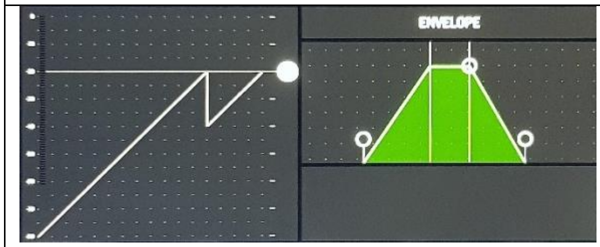
```
0 “mdl”:   AP2
1 “f”:  logf [100, 10000, 100] Hz, freq
2 “q”:  logf [.442, 10, 181] q
```

Gate plugins



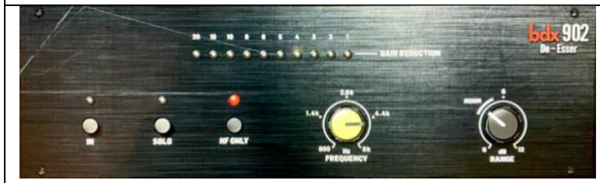
Standard Gate/Expander

```
0 “mdl”:   GATE
1 “thr”:   linf [-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”:   linf [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
```



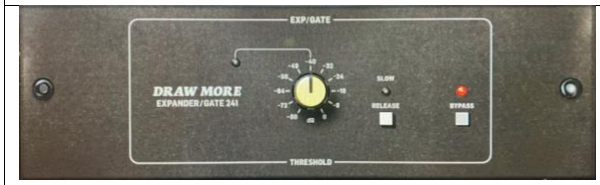
Standard Ducker

```
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
3 “hold”:  linf [1, 200, 200] ms, hold
5 “rel”:   linf [20, 4000, 130] ms, release
```



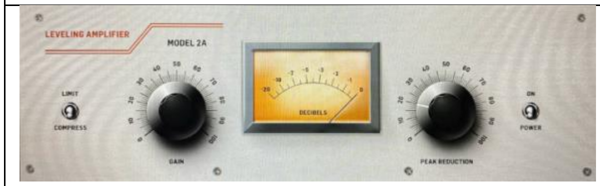
DBX 902 DeEsser

```
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
```



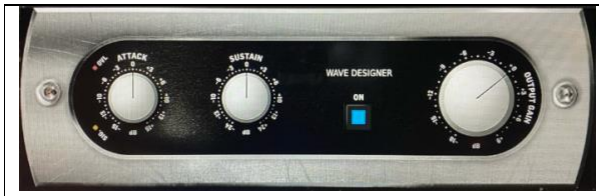
DrawMore Expander Gate 241

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



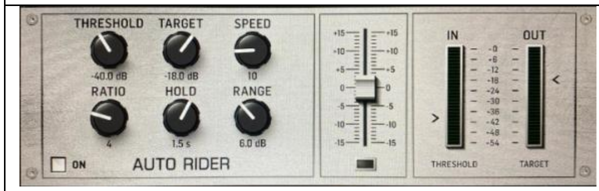
Leveling Amplifier 2A

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



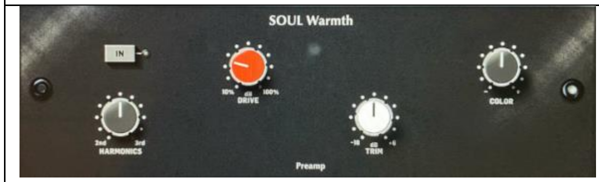
Wave Designer

```
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
```



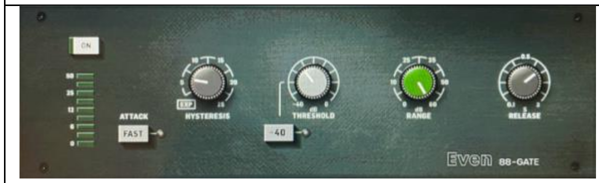
Auto Rider Dynamics

```
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
```



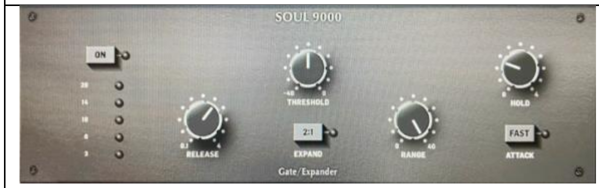
Soul Warmth Preamp

```
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
```



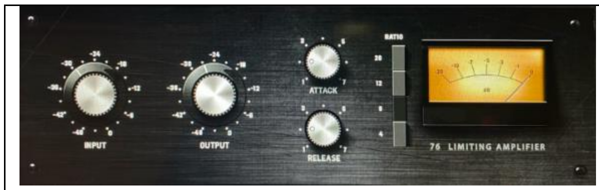
Even 88-Gate

```
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
```



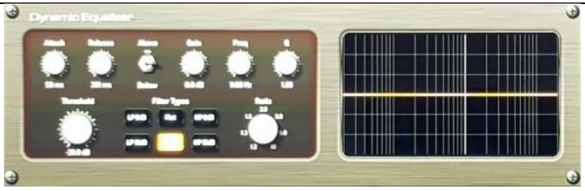
SSL 9000 Channel Gate

```
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
```



76 Limiter Amp

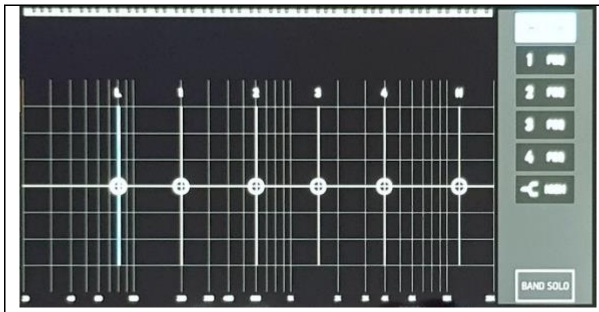
```
 0 “mdl”:   76LA
 1 “in”:    linf [-48, 0, 97] dB, input
 2 “out”:   linf [-48, 0, 97] dB
 3 “att”:   linf [1, 7, 61] attack
 4 “rel”:   linf [1, 7, 61] release
 5 “ratio”: str [4, 8, 12, 20, ALL] ratio
```



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 “g”:     logf [.442, 10, 181] q
 9 “mode”:  str [low, high] mode
```

EQ plugins



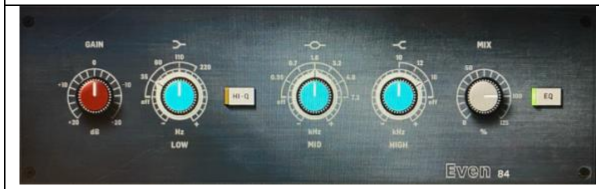
Standard EQ

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
```

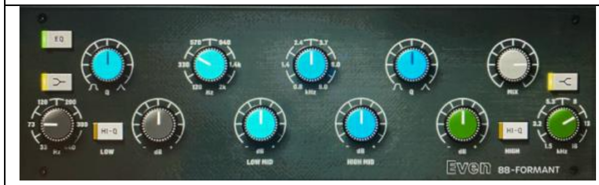
Bus, mtx, main:

```
 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
```



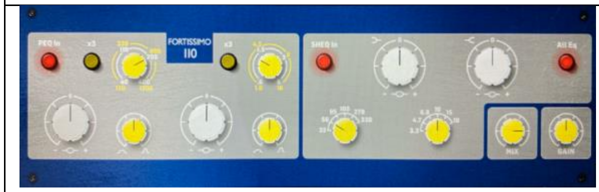
Even 84 EQ

```
 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
```



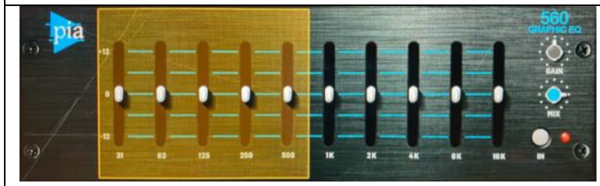
Even 88-Formant EQ

```
 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
```



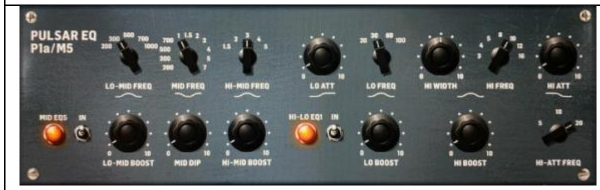
Focusrite ISA 110 EQ

```
 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
 4 “lmq”:   linf [0, 10, 101] lm q
 5 “lmf3”:  int [0, 1] lm /3
 6 “hmf”:   linf [0, 10, 101] hm freq
 7 “hmg”:   linf [-5, 5, 101] hm gain
 8 “hmq”:   linf [0, 10, 101] hm q
 9 “hmf3”:  int [0, 1] hm x3
10 “shv”:   inf [0, 1] shv on
11 “lf”:    str [33, 56, 95, 160,
                 270, 460] lf freq
11 “lg”:    linf [-5, 5, 101] lf gain
13 “hf”:    str [3k3, 4k7,6k8, 10k,
                 15k, 18k] hf freq
14 “hg”:    linf [-5, 5, 101] hf q
15 “g”:     linf [-18, 18, 73] gain
```



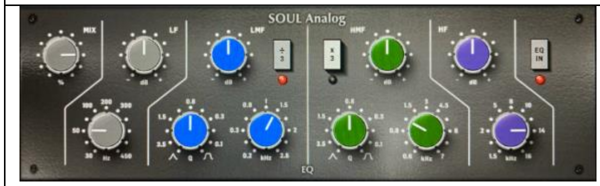
PIA 560 EQ

```
 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
```



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

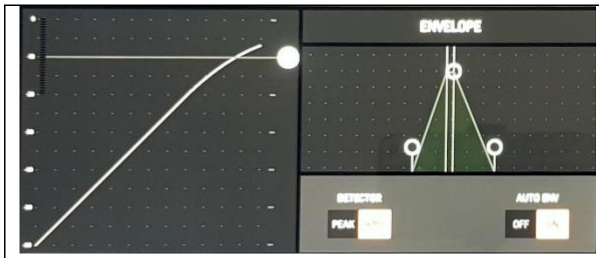


Soul Analog EQ

```
 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
```

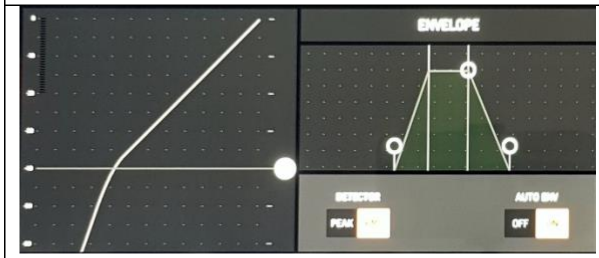
	*12 **"hf"**: linf [0, 10, 101] hf freq* *13 **"hg"**: linf [-5, 5, 101] hf gain*

Compressor plugins



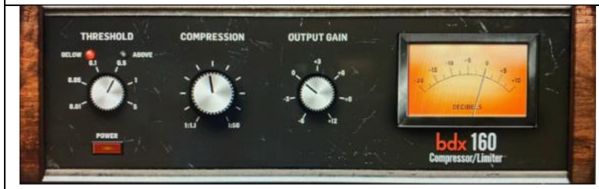
Standard compressor

```
 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
 4 “knee”:  int [0…5] knee
 5 “det”:   str [PEAK, RMS] detector
 6 “att”:   linf [0, 120, 121] ms, attack
 7 “hld”:   linf [1, 200, 200] ms, hold
 8 “rel”:   logf [4, 4000, 130] ms release
 9 “env”:   str [LIN, LOG] envelope
10 “auto”:  int [0, 1] auto
```



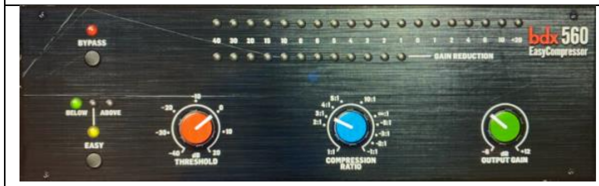
Standard expander

```
 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
 4 “knee”:  int [0…5] knee
 5 “det”:   str [PEAK, RMS] detector
 6 “att”:   linf [0, 120, 121] ms, attack
 7 “hld”:   linf [1, 200, 200] ms, hold
 8 “rel”:   logf [4, 4000, 130] ms release
 9 “env”:   str [LIN, LOG] envelope
10 “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
```

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
```

Fairkid Model 670

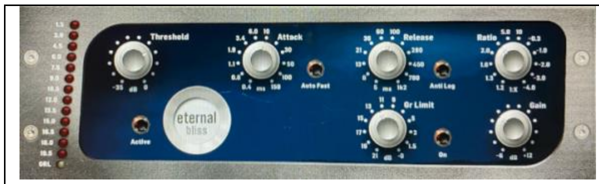
```
 0 “mdl”:   F670
 1 “mix”:   linf [0, 100, 101] %, mix
 2 “gain”:  linf [-6, 12, 37] dB, gain
 3 “in”:    linf [-20, 0, 81] dB, input
 4 “thr”:   linf [0, 10, 41] thr
 5 “time”:  int [1…6] time
 6 “bias”:  linf [0, 1, 101] bias
```

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
```

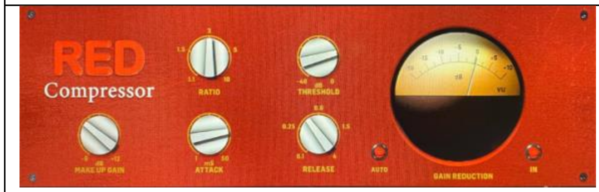
No Stressor

```
 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
 5 “ratio”: str [1.5:1, 2:1, 3:1, 4:1, 6:1,
                10:1, 20:1, NUKE] ratio
```



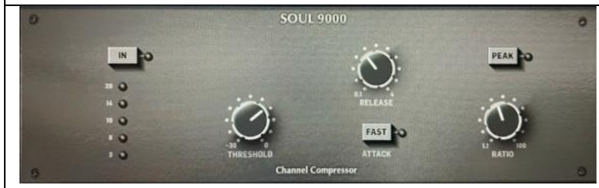
Eternal Bliss

```
 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
```



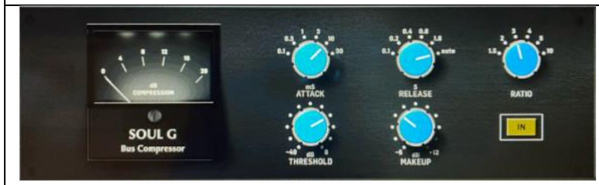
Red Compressor

```
 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
 7 “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

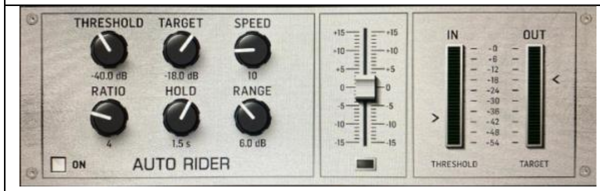
```
 0 “mdl”:   SBUS
 1 “mix”:   linf [0, 100, 101] %, mix
 2 “gain”:  linf [-6, 12, 37] dB, gain
 3 “thr”:   linf [-48, 0, 81] dB, thr
 4 “ratio”: flt [1.5, 2.0, 3.0, 4.0, 5.0,
                10.0] ratio
 5 “att”:   flt [0.1, 0.3, 1.0, 3.0, 10.0,
                30.0] ratio
 6 “rel”:   str [0.1, 0.2,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

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
```

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
```



Auto Rider Dynamics

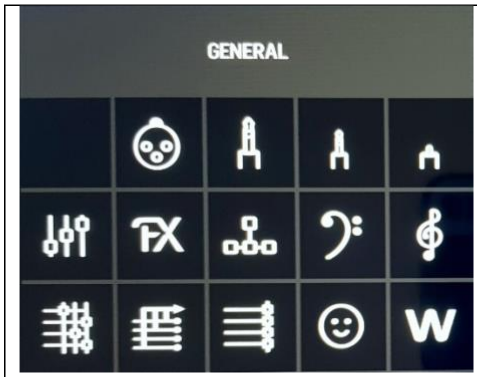
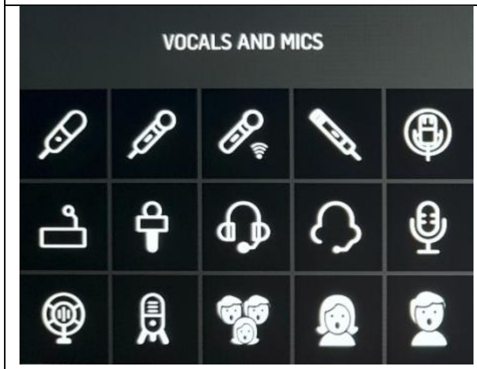
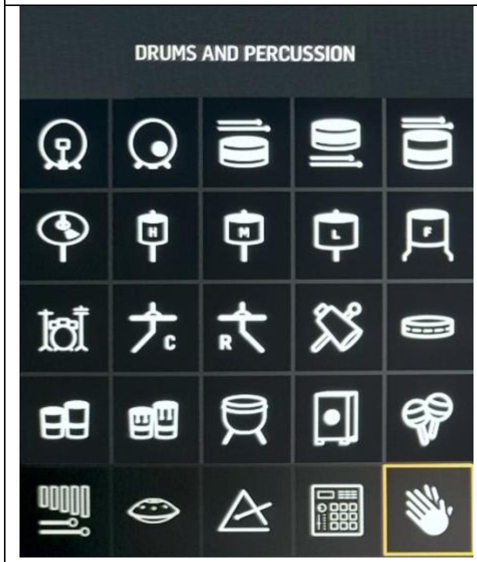
```
0 “mdl”:   RIDE
1 “mix”:   linf [0, 100, 101] %, mix
2 “gain”:  linf [-6, 12, 37] dB, gain
3 “thr”:   linf [-54, 18, 73] dB, thr
4 “tgt”:   linf [-48, 0, 97] dB, target
5 “spd”:   int [1…50] speed
6 “ratio”: flt [2.0, 4.0, 8.0,
                20.0, 100.0] ratio
7 “hld”:   logf [.1, 10, 65] s, hold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

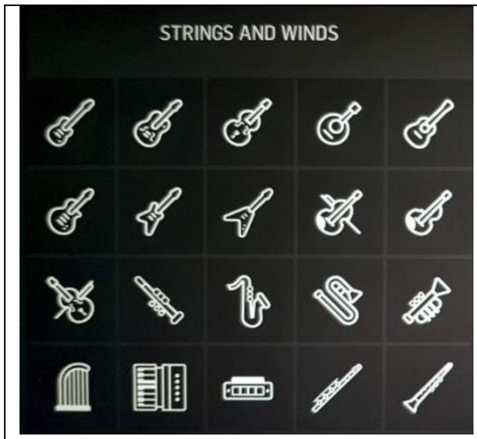
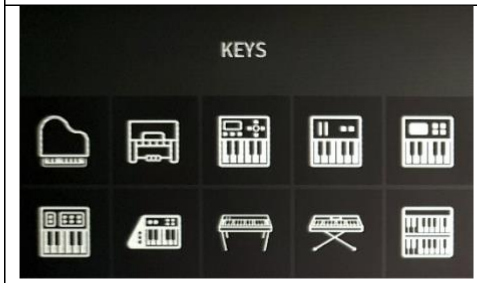
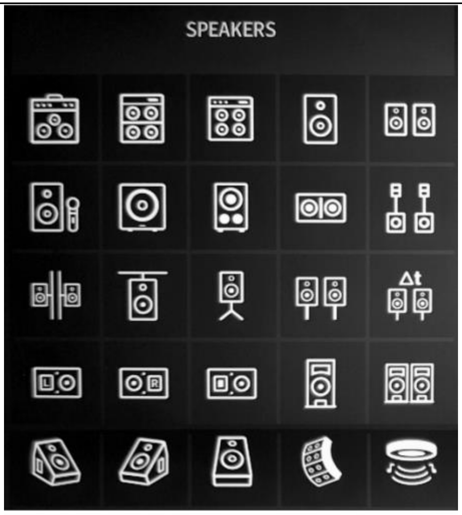
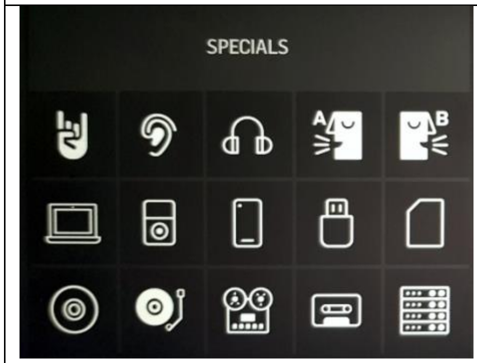
Draw More Compressor

```
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
```

Appendix: WING Icons

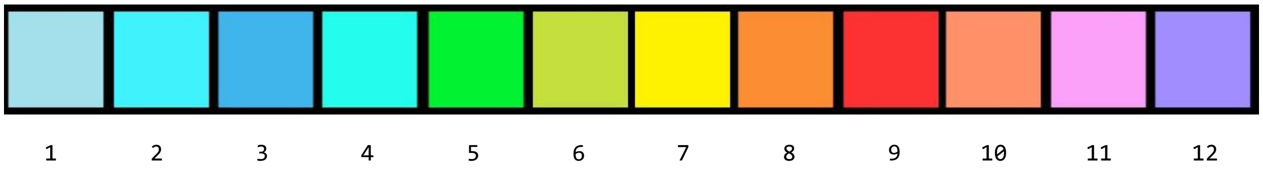
The table below gives the list of icons available with WING. The icon numbers are listed to the right of the icons.

	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 Snapshot and JSON Data Structure:

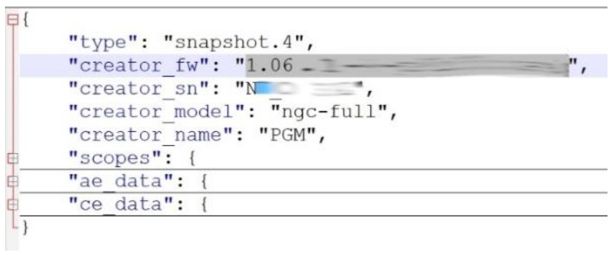
A WING snapshot (also called Snapfile when saved to a file) is organized as a collection of classes, sub-classes and objects regrouping attributes and values in logical groups. These can be represented as a hierarchical tree. A JSON[10] notation is used to describe and store the hierarchical tree.

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



```
{
    "type": "snapshot.4",
    "creator_fw": "1.06[redacted]",
    "creator_sn": "N[redacted]",
    "creator_model": "ngc-full",
    "creator_name": "PGM",
    "scopes": {
    "ae_data": {
    "ce_data": {
}
```

Descriptionn

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.

```
“type”: string,
“creator_fw”: string,
“creator_sn”: string,
“creator_model”: string,
“creator_name”: string,
```

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

scopes

scopes: A large set of *Boolean* {*true|false*} values to list what has been saved at snapshot time that can also be used as a reminder of the initial purpose of the snapshot. This set of values is also used at load time to show what console parameter groups will be affected by the recall operation in adjusting what should be loaded when recalling a scene.

The scopes class contains the following objects:

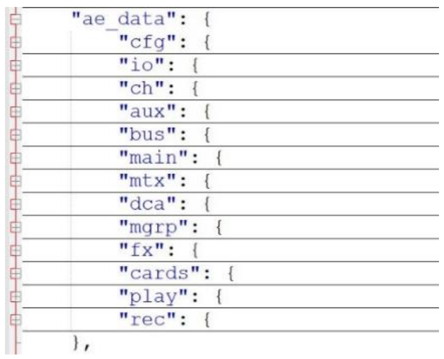
ch, aux, bus, main, mtx, fx, routin, routout, cfg, area, data, with:

```
"scopes": {
        "ch": {
                "1": Boolean
                …
                "40": Boolean
        }
        "aux": {
                "1": Boolean
                …
                "8": Boolean
        }
        "bus": {
                "1": Boolean
                …
                "16": Boolean
        }
        "main": {
                "1": Boolean
                …
                "4": Boolean
        }
        "mtx": {
                "1": Boolean
                …
                "8": Boolean
        }
        "fx": {
                "1": Boolean
                …
                "16": Boolean
        }
        "routin": {
                "1": Boolean
                …
                "13": Boolean
        }
        "routout": {
                "1": Boolean
                …
                "11": Boolean
        }
        "cfg": {
                "groups": Boolean
                "audio": Boolean
                "surface": Boolean
        }
        "area": {
                "L": Boolean
                "C": Boolean
                "R": Boolean
        }
        ”data”{
                “1”: Boolean
                …
                “9”: Boolean
        }
}
```

Scopes are not elements that can be programmatically changed. They are only set at snapshot time using the console main LCD. As mentioned earlier, they are used at save time to notify what was targeted for update, and at restore time on the console, to indicate what will be modified as the snapshot is restored to the desk.

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:



```
"ae_data": {
    "cfg": {
    "io": {
    "ch": {
    "aux": {
    "bus": {
    "main": {
    "mtx": {
    "dca": {
    "mgrp": {
    "fx": {
    "cards": {
    "play": {
    "rec": {
},
```

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 later in this document) as the JSON structure parameters is a key subset of the tokens used by the API for `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.

The ae_data class contains the following objects: ***cfg, io, ch, aux, bus, main, mtx, dca, mgrp, fx, cards, play, rec***, shown in the following pages using the notation conventions above.

```
"ae_data": {
        "cfg": {
                "clkrate": 48000,
                "clksrc": "INT",
                "mainlink": Boolean
                "dcamgrp": Boolean
                "usbacfg": "2/2",
                "mon": {
                        "1": {
                                "inv": Boolean
                                "pan": 0,
                                "wid": 100,
                                "eq": {
                                        "on": Boolean
                                        "lsg": 0,
                                        "lsf": 60.13884,
                                        "1g": 0,
                                        "1f": 129.8763,
                                        "1q": 1.995882,
                                        …
                                        "6g": 0,
                                        "6f": 6013.884,
                                        "6q": 1.995882,
                                        "hsg": 0,
                                        "hsf": 11999.27
                                },
                                "lim": 0,
                                "dly": {
                                        "on": Boolean
                                        "m": 0.1
                                },
                                "dim": 20,
                                "srclvl": 0,
                                "src": "MAIN.1"
                        },
                        "2": {},
                }
                "solo": {
                        "mode": "LIVE",
                        "mon": "A",
                        "mute": Boolean
                        "chtap": "PFL",
                        "bustap": "AFL",
                        "maintap": "PFL",
                        "mtxtap": "PFL"
```

```
            },
            “rta”: {
                    “dec”: “MED”,
                    “det”: “RMS”,
                    “range”: 30,
                    “g”: 0,
                    “auto”: true
            },
            “talk”: {
                    “assign”: “OFF”,
                    “A”: {
                            “mode”: “AUTO”,
                            “mondim”: Boolean
                            “busdim”: 0,
                            “B1”: Boolean
                            …
                            “B16”: Boolean
                            “M1”: Boolean
                            …
                            “M4”: Boolean
                    },
                    “B”: {}
            },
            “osc”: {
                    “1”: {
                            “lvl”: -6,
                            “mode”: “SINE”,
                            “f”: 999.992
                    },
                    “2”: {}
            },
            “gpio”: {
    }
    “io”: {
            “altsw”: Boolean
            “in”: {
                    “LCL”: {
                            “1”: {
                                    “mode”: “M”,
                                    “g”: 0,
                                    “vph”: Boolean
                                    “mute”: Boolean
                                    “col”: 1,
                                    “name”: string
                                    “icon”: 0,
                                    “tags”: string
                            }
                            “2”…“8”: {}
                    }
                    “AUX”: {}
                    “A”: {
                            “1”: {
                                    “mode”: “M”,
                                    “g”: 0,
                                    “vph”: Boolean
                                    “mute”: Boolean
                                    “col”: 1,
                                    “name”: string
                                    “icon”: 0,
                                    “tags”: string
                            },
                            “2”…“48”: {}
                    }
                    “B”: {}
                    “C”: {}
                    “SC”: {
                            “1”: {
                                    “mode”: “M”,
                                    "g": 0,
                                    "vph": Boolean
                                    "mute": Boolean
                                    "col": 1,
                                    "name": string
                                    "icon": 0,
                                    "tags": string
                            },
                            "2"…"32": {}
                    }
                    "USB": {
                            "1": {
                                    "mode": "ST",
                                    "g": 0,
                                    "vph": Boolean
                                    "mute": Boolean
                                    "col": 8,
                                    "name": "USB 1/2",
                                    "icon": 605,
                                    "tags": string
                            },
                            "2"…"48": {}
                    }
                    "CRD": {
                            "1": {
                                    "mode": "M",
                                    "g": 0,
                                    "vph": Boolean
                                    "mute": Boolean
                                    "col": 1,
                                    "name": string
                                    "icon": 0,
                                    "tags": string
                            },
                            "2"…"64": {}
                    }
                    "MOD": {}
                    "PLAY": {
                            "1": {
                                    "mode": "M",
                                    "g": 0,
                                    "vph": Boolean
                                    "mute": Boolean
                                    "col": 1,
                                    "name": string
                                    "icon": 0,
                                    "tags": string
                            },
                            "2"…"4": {}
                    }
                    "AES": {
                            "1": {
                                    "mode": "M",
                                    "g": 0,
                                    "vph": Boolean
                                    "mute": Boolean
                                    "col": 1,
                                    "name": string
                                    "icon": 0,
                                    "tags": string
                            },
                            "2": {}
                    }
                    "USR": {
                            "1": {
                                    "mode": "M",
                                    "g": 0,
                                    "vph": Boolean
```

```
                                “mute”: Boolean
                                “col”: 1,
                                “name”: string
                                “icon”: 0,
                                “tags”: string
                        },
                        “2”…“24”: {}
                }
                “OSC”: {
                        “1”: {
                                “mode”: “M”,
                                “g”: 0,
                                “vph”: Boolean
                                “mute”: Boolean
                                “col”: 1,
                                “name”: string
                                “icon”: 0,
                                “tags”: string
                        },
                        “2”: {}
                }
        }
        “out”: {
                “LCL”: {
                        “1”: {
                                “grp”: “BUS”,
                                “in”: 1
                        },
                        …
                        “8”: {}
                }
                “AUX”: {}
                “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”: {}
                “REC”: {
                        “1”: {
                                “grp”: “OFF”,
                                “in”: 1
                        },
                        "2"…"4": {}
                }
                "AES": {
                        "1": {
                                "grp": "OFF",
                                "in": 1
                        },
                        "2": {}
                }
        }
        "user": {
                "1": {
                        "grp": "OFF",
                        "in": 1,
                        "tap": "PRE",
                        "lr": "L+R"
                }
                "2"…"24": {}
        }
}
"ch": {
        "1": {
                "in": {
                        "set": {
                                "srcauto": Boolean
                                "altsrc": Boolean
                                "inv": Boolean
                                "trim": 0,
                                "bal": 0,
                                "dly": 0
                        },
                        "conn": {
                                "grp": "LCL",
                                "in": 1,
                                "altgrp": "OFF",
                                "altin": 1
                        }
                },
                "flt": {
                        "lc": Boolean
                        "lcf": 100.2375,
                        "hc": Boolean
                        "hcf": 10023.74,
                        "tf": Boolean
                        "mdl": "TILT",
                        "tilt": 0
                },
                "col": 1,
                "name": string
                "icon": 1,
                "led": Boolean
                "mute": Boolean
                "fdr": -144,
                "pan": 0,
                "wid": 100,
                "solosafe": Boolean
                "mon": "A",
                "proc": "GEDI",
                "ptap": "4",
                "peq": {
                        "on": Boolean
                        "1g": 0,
                        "1f": 99.68543,
                        "1q": 1.995882,
                        …
                        "3g": 0,
                        "3f": 10016.53,
                        "3q": 1.995882
```

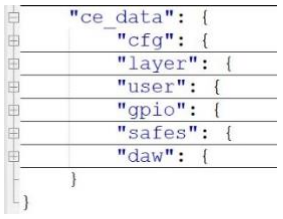
```
},
"gate": {
        "on": Boolean
        "mdl": "GATE",
        "thr": -40,
        "range": 40,
        "att": 10,
        "hld": 10,
        "rel": 199.4043,
        "acc": 0,
        "ratio": "1:3"
},
"gatesc": {
        "type": "OFF",
        "f": 1002.374,
        "q": 1.995882,
        "src": "SELF",
        "tap": "IN"
},
"eq": {
        "on": Boolean
        "mdl": "STD",
        "mix": 100,
        "lg": 0,
        "lf": 80.19642,
        "lq": 1.995882,
        "leq": "SHV",
        "1g": 0,
        "1f": 200,
        "1q": 1.995882,
        …
        "4g": 0,
        "4f": 3990.524,
        "4q": 1.995882,
        "hg": 0,
        "hf": 11999.27,
        "hq": 1.995882,
        "heq": "SHV"
},
"dyn": {
        "on": Boolean
        "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": Boolean
```

```
                        “ins”: “NONE”
                },
                “main”: {
                        “1”: {
                                “on”: Boolean
                                “lvl”: 0
                        },
                        “2”…“4”: {}
                },
                “send”: {
                        “1”: {
                                “on”: Boolean
                                “lvl”: -144,
                                “pon”: Boolean
                                “ind”: Boolean
                                “mode”: “PRE”,
                                “plink”: Boolean
                                “pan”: 0,
                                “wid”: 100
                        },
                        “2”…“16”: {}
                },
                “postins”: {
                        “on”: Boolean
                        “mode”: “FX”,
                        “ins”: “NONE”,
                        “w”: 0
                },
                “tags”: string
        },
        “2”…“40”: {}
}
“aux”: {
        “1”: {
                “in”: {
                        “set”: {
                                “srcauto”: Boolean
                                “altsrc”: Boolean
                                “inv”: Boolean
                                “trim”: 0,
                                “bal”: 0
                        },
                        “conn”: {
                                “grp”: “USB”,
                                “in”: 1,
                                “altgrp”: “OFF”,
                                “altin”: 1
                        }
                },
                “col”: 8,
                “name”: “USB”,
                “icon”: 605,
                “led”: Boolean
                “mute”: Boolean
                “fdr”: -144,
                “pan”: 0,
                “wid”: 100,
                “solosafe”: Boolean
                “mon”: “A”,
                “eq”: {
                        “on”: Boolean
                        “mix”: 100,
                        “lg”: 0,
                        “lf”: 80.19642,
                        “lq”: 1.995882,
                        “leq”: “SHV”,
                        “1g”: 0,
                        “1f”: 399.0524,
                        "1q": 1.995882,
                        "2g": 0,
                        "2f": 2499.799,
                        "2q": 1.995882,
                        "hg": 0,
                        "hf": 11999.27,
                        "hq": 1.995882,
                        "heq": "SHV"
                },
                "preins": {
                        "on": Boolean
                        "ins": "NONE"
                },
                "main": {
                        "1": {
                                "on": Boolean
                                "lvl": 0
                        },
                        "2"…"4": {}
                },
                "send": {
                        "1": {
                                "on": Boolean
                                "lvl": -144,
                                "pon": Boolean
                                "ind": Boolean
                                "mode": "PRE",
                                "plink": Boolean
                                "pan": 0,
                                "wid": 100
                        },
                        "2"…"16": {}
                },
                "tags": string
        },
        "2"…"8": {}
}
"bus": {
        "1": {
                "in": {
                        "set": {
                                "inv": Boolean
                                "trim": 0,
                                "bal": 0
                        }
                },
                "col": 1,
                "name": string
                "icon": 0,
                "led": Boolean
                "busmono": Boolean
                "mute": Boolean
                "fdr": -144,
                "pan": 0,
                "wid": 100,
                "mon": "A" | "B"
                "busmode": "PRE",
                "eq": {
                        "on": Boolean
                        "mdl": "STD",
                        "mix": 100,
                        "lg": 0,
                        "lf": 60.13884,
                        "lw": 0.99797,
                        "leq": "SHV",
                        "1g": 0,
                        "1f": 129.8763,
                        "1w": 0.99797,
                        …
                        "6g": 0,
                        "6f": 6013.884,
                        "6w": 0.99797,
                        "hg": 0,
                        "hf": 11999.27,
                        "hq": 0.99797,
                        "heq": "SHV",
                        "tilt": 0
                },
                "dyn": {
                        "on": Boolean
                        "mdl": "COMP",
                        "mix": 100,
                        "gain": 0,
                        "thr": -10,
                        "ratio": 3,
                        "knee": 3,
                        "det": "RMS",
                        "att": 50,
                        "hld": 20,
                        "rel": 152.5652,
                        "env": "LOG",
                        "auto": Boolean
                },
                "dynxo": {
                        "depth": 6,
                        "type": "OFF",
                        "f": 1002.374
                },
                "dynsc": {
                        "type": "OFF",
                        "f": 1002.374,
                        "q": 1.995882,
                        "src": "SELF",
                        "tap": "BUS"
                },
                "preins": {
                        "on": Boolean
                        "ins": "NONE"
                },
                "main": {
                        "1": {
                                "on": Boolean
                                "lvl": 0
                        },
                        "2"…"4": {}
                },
                "send": {
                        "1": {
                                "on": Boolean
                                "lvl": -144
                        },
                        "2"…"8": {}
                },
                "postins": {
                        "on": Boolean
                        "ins": "NONE"
                },
                "tags": string
        },
        "2"…"16": {}
}
"main": {
        "1": {
                "in": {
                        "set": {
                                "inv": Boolean
                                “trim”: 0,
                                “bal”: 0
                        }
                },
                “col”: 1,
                “name”: string
                “icon”: 509,
                “led”: Boolean
                “busmono”: Boolean
                “mute”: Boolean
                “fdr”: -144,
                “pan”: 0,
                “wid”: 100,
                “mon”: “A” | “B”
                “eq”: {
                        “on”: Boolean
                        “mdl”: “STD”,
                        “mix”: 100,
                        “lg”: 0,
                        “lf”: 60.13884,
                        “lw”: 0.99797,
                        “leq”: “SHV”,
                        “1g”: 0,
                        “1f”: 129.8763,
                        “1w”: 0.99797,
                        …
                        “6g”: 0,
                        “6f”: 6013.884,
                        “6w”: 0.99797,
                        “hg”: 0,
                        “hf”: 11999.27,
                        “hq”: 0.99797,
                        “heq”: “SHV”,
                        “tilt”: 0
                },
                “dyn”: {
                        “on”: Boolean
                        “mdl”: “COMP”,
                        “mix”: 100,
                        “gain”: 0,
                        “thr”: -10,
                        “ratio”: 3,
                        “knee”: 3,
                        “det”: “RMS”,
                        “att”: 50,
                        “hld”: 20,
                        “rel”: 152.5652,
                        “env”: “LOG”,
                        “auto”: Boolean
                },
                “dynxo”: {
                        “depth”: 6,
                        “type”: “OFF”,
                        “f”: 1002.374
                },
                “dynsc”: {
                        “type”: “OFF”,
                        “f”: 1002.374,
                        “q”: 1.995882,
                        “src”: “SELF”,
                        “tap”: “BUS”
                },
                “preins”: {
                        “on”: Boolean
                        “ins”: “NONE”
                },
                “send”: {
                        “1”: {
```

```
                                        "on": Boolean
                                        "lvl": 0
                                },
                                …
                                "8": {}
                        },
                        "postins": {
                                "on": Boolean
                                "ins": "NONE"
                        },
                        "tags": string
                },
                "2"…"4": {}
        }
        "mtx": {
                "1": {
                        "in": {
                                "set": {
                                        "inv": Boolean
                                        "trim": 0,
                                        "bal": 0
                                }
                        },
                        "dir": {
                                "1": {
                                        "on": Boolean
                                        "lvl": -144,
                                        "inv": Boolean
                                        "in": "OFF",
                                        "tap": "PRE"
                                },
                                "2": {}
                        },
                        "col": 1,
                        "name": string
                        "icon": 0,
                        "led": Boolean
                        "busmono": Boolean
                        "mute": Boolean
                        "fdr": -144,
                        "pan": 0,
                        "wid": 100,
                        "mon": "A" | "B"
                        "eq": {
                                "on": Boolean
                                "mdl": "STD",
                                "mix": 100,
                                "lg": 0,
                                "lf": 60.13884,
                                "lw": 0.99797,
                                "leq": "SHV",
                                "1g": 0,
                                "1f": 129.8763,
                                "1w": 0.99797,
                                …
                                "6g": 0,
                                "6f": 6013.884,
                                "6w": 0.99797,
                                "hg": 0,
                                "hf": 11999.27,
                                "hq": 0.99797,
                                "heq": "SHV",
                                "tilt": 0
                        },
                        "dyn": {
                                "on": Boolean
                                "mdl": "COMP",
                                "mix": 100,
                                "gain": 0,
                                "thr": -10,
                                "ratio": 3,
                                "knee": 3,
                                "det": "RMS",
                                "att": 50,
                                "hld": 20,
                                "rel": 152.5652,
                                "env": "LOG",
                                "auto": Boolean
                        },
                        "dynxo": {
                                "depth": 6,
                                "type": "OFF",
                                "f": 1002.374
                        },
                        "dynsc": {
                                "type": "OFF",
                                "f": 1002.374,
                                "q": 1.995882,
                                "src": "SELF",
                                "tap": "BUS"
                        },
                        "preins": {
                                "on": Boolean
                                "ins": "NONE"
                        },
                        "postins": {
                                "on": Boolean
                                "ins": "NONE"
                        },
                        "dly": {
                                "on": Boolean
                                "m": 0.1
                        },
                        "tags": string
                },
                "2"…"8": {}
        }
        "dca": {
                "1": {
                        "name": string
                        "col": 1
                        "icon": 0
                        "led": Boolean
                        "mute": Boolean
                        "fdr": -144,
                        "mon": "A" | "B"
                },
                "2"…"8": {}
       }
       "mgrp": {
                "1": {
                        "name": string
                        "mute": Boolean
                }
                "2"…"8": {}
       }
       "fx": {
                "1": {
                        "mdl": "NONE",
                        "fxmix": 100
                }
                "2"…"16": {}
       }
       "cards": {
                "wlive": {
                        "mode": "IND",
                        “sorting”: “DATE”,
                        “autoin”: “OFF”,
                        “1”: {
                                “cfg”: {
                                        “rectracks”: “32”,
                                        “playmode”: “PLAY”
                                }
                        },
                        “2”: {}
                }
        }
        “play”: {}
        “rec”: {
                “path”: string
                “resolution”: 16 | 24
                “channels”: 1
        }
}
```

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***, as shown below:



```
    "ce_data": {
        "cfg": {
        "layer": {
        "user": {
        "gpio": {
        "safes": {
        "daw": {
    }
}
```

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.

```
"$ctl": {
        "cfg": {
                "lights": {
                        "btns": 10,
                        "leds": 90,
                        "meters": 40,
                        "rgbleds": 50,
                        "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": Boolean
                }
                "mtrfsc": {
                        "in": "PRE",
                        "bus": "POST",
                        "main": "POST",
                        "mtx": "POST",
                        "dca": "PRE"
                }
                "mtrpage": {
                        "in": "PRE",
                        "bus": "POST",
                        "main": "POST",
                        "mtx": "POST",
                        "dca": "PRE"
                }
                "mainmtr": string,
                "mainpos": "AUTO",
                "soloexcl": Boolean,
                "selfsolo": Boolean,
                "solofsel": Boolean,
                "sof2solo": Boolean,
                "layerlinkl": Boolean,
                "layerlinkr": Boolean,
                "autoview": Boolean,
                "csctouch": Boolean,
                "autosel_L": Boolean,
                "autosel_C": Boolean,
                "autosel_R": Boolean,
                "fdrrsel": Boolean,
                "fdrres": "AUTO",
                "fdrspd": "MED",
                "soffdr": "L/C",
                "srcdisp": Boolean,
                "lockmtr": Boolean,
                "timefmt": "12H" | "24H",
                "datefmt": "YMD",
                "filesort": "A->Z"
        }
        "layer": {
                "L" : {
                        "sel": 1
                        "1": {
                                "ofs": 0
                                "name": "CH1-12" | "CH13-24" | "CH25-36" | "CH37-AUX" |
                                         "BUSES" | "USER1" | "USER2"
                                "1": {
                                        "type": "CH",
                                        "i": 1,
                                        "dst": 1
                                }
                                …
                                "24": {}
                        }
                        "2"…"7": {}
                }
                "C": {
                        "sel": 4
                        "1": {
                                "ofs": 0
                                "name": "DCA" | "AUX" | "BUSES" | "USER1" | "USER2"
                                "1": {
                                        "type": "OFF" | "DCA"
                                        "i": 1,
                                        "dst": 1
                                }
                                …
                                "16": {}
                        }
                        "2"…"6": {}
                }
                "R": {
                        "sel": 1
                        "1": {
                                "ofs": 0
                                "name": "MAIN" | "DCA" | "CH1-40" | "AUX" | "BUSES" |
                                         "USER1" | "USER2"
                                "1": {
                                        "type": "OFF" | "BUS" | "DCA" | "CHI" |
                                        "i": 1,
                                        "dst": 1
                                }
                                …
                                "40": {}
                }
                “2”…”7”: {}
            }
    }
    “user”: {
            “sel”: 1
            “mode”: “USER”
            “cmode”: “HA”
            “usermode”: “BUS”
            “tapflash”: “ON”
            “gpio”: {
                “1”: {
                    “bu”: {
                        “mode”: “OFF”,
                        “name”: “GPIO 1”
                    }
                },
                “2”…”4”: {}
            }
            “user”: {
                “1”: {
                    “led”: Boolean,
                    “col”: 1,
                    “enc”: {
                        “mode”: “OFF”,
                        “name”: string
                    },
                    “bu”: {
                        “mode”: “OFF”,
                        “name”: string
                    },
                    “bd”: {
                        “mode”: “OFF”,
                        “name”: string
                    }
                }
                “2”…”3”: {}
            }
            “daw1”: {
                “1”: {
                    “bu”: {
                        “mode”: “OFF”
                        “name”: string
                        “btn”: string
                    }
                    “bd”: {
                        “mode”: “OFF”
                        “name”: string
                        “btn”: string
                    }
                }
                “2”…”4”: {}
            }
            “daw2…daw4”: {}
            “1”: {
                “1”: {
                    “led”: Boolean,
                    “col”: 1,
                    “enc”: {
                        “mode”: “DCA”,
                        “name”: string
                        “dca”: string
                    },
                    “bu”: {
                        “mode”: “OFF”,
                        “name”: string
                    },
                    “bd”: {
```

```
                                “mode”: “OFF”,
                                “name”: string
                        }
                }
                “2”…”4”{}
        }
        “2”…”16”: {}
        “cuser”: {
                “1”: 1
                “2”: 1
                “3”: 1
        }
}
“gpio”: {
        “1”: {
                “mode”: “TGLNO”,
                “gpstate”: false
        },
        “2”…”4”: {}
}
“safes”: {
        “ch”: {
                “1”: Boolean
                …
                “40”: Boolean
        }
        “aux”: {
                “1”: Boolean
                …
                “8”: Boolean
        }
        “bus”: {
                “1”: Boolean
                …
                “16”: Boolean
        }
        “main”: {
                “1”: Boolean
                …
                “4”: Boolean
        }
        “mtx”: {
                “1”: Boolean
                …
                “8”: Boolean
        }
        “fx”: {
                “1”: Boolean
                …
                “16”: Boolean
        }
        “routin”: {

                “1”: Boolean
                …
                “13”: Boolean
        }
        “routout”: {
                “1”: Boolean
                …
                “11”: Boolean
        }
        “cfg”: {
                “group”: Boolean
                “audio”: Boolean
                 “surface”: Boolean
        }
        “area”: {
```

```
                                “L”: Boolean
                                “C”: Boolean
                                “R”: Boolean
                        }
                        “data”: {
                                “1”: Boolean
                                …
                                “9”: Boolean
                        }
                }
                “daw”: {
                        “on”: true,
                        “conn”: “USB”,
                        “emul”: “MCU”,
                        “config”: “MSTR2EXT”,
                        “ccup”: Boolean,
                        “preset”: “-”
                }
                “midi”: {
                        “enchctl”: “OFF”,
                        “enfxctl”: “OFF”,
                        “encustctl”: “OFF”,
                        “ensysex”: “USB”
                }
                “OSC”: {
                        “ronly”: Boolean
                }

        }
```

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.