VSKP

Violet Simple Key Protocol

A line-based text protocol for querying and controlling mixer parameters over TCP

Protocol Reference

Table of Contents

1. Introduction

VSKP (Violet Simple Key Protocol) is a line-based text protocol for querying and controlling mixer parameters over TCP. It is designed for simplicity: connect with any TCP client (telnet, netcat, a custom script), send plain-text commands, and receive plain-text responses. No handshake or authentication is required.

Default port: 1703

2. Connection

Open a TCP connection to the VSKP port. There is no banner or handshake — the server is immediately ready to accept commands.

```
$ nc 192.168.1.10 1703
?i.0.mix
i.0.mix=-10
```

Lines are delimited by `\n` (LF). Carriage returns (`\r`) are stripped. Each line is one command; each response is one line.

2.1 Timeout

By default there is no idle timeout. You can set one with:

```
tcptimeout=30
```

This closes the connection after 30 seconds of inactivity. Set to `0` to disable. The `noop` command resets the timeout timer without performing any action.

2.2 Disconnecting

Send `quit`, `exit`, or `bye` to cleanly close the connection.

3. Key Naming

Mixer parameters are addressed by dot-separated key paths. The first component identifies the channel type, the second the zero-indexed channel number, and the rest describes the parameter.

Prefix	Type	Example Key	Description
`i`	Input	`i.0.mix`	Input 1 fader level
`a`	Aux	`a.3.mute`	Aux 4 mute
`s`	Subgroup	`s.0.pan`	Sub 1 pan
`f`	FX Send	`f.1.mix`	FX 2 send level
`r`	FX Return	`r.0.mix`	FX Return 1 level
`v`	VCA	`v.0.mix`	VCA 1 fader level
`x`	Matrix	`x.0.mix`	Matrix 1 fader level
`m`	Master	`m.mix`	Master fader
`h`	Preamp	`h.0.gain`	Preamp 1 gain
`d`	DAW Return	`d.0.mix`	DAW Return 1 level
`t`	Talkback	`t.mix`	Talkback level
`osc`	Oscillator	`osc.freq`	Oscillator frequency
`out`	Output	`out.0.gain`	Output 1 gain

3.1 Common Parameter Suffixes

Suffix	Type	Description
`.mix`	numeric	Fader level (dB, mix-scaled)
`.mute`	numeric	Mute (0 = unmuted, 1 = muted)
`.pan`	numeric	Pan position (0.0–1.0)
`.solo`	numeric	Solo (0/1)
`.name`	text	Channel name
`.color`	numeric	Channel colour index
`.eq.b1.freq`	numeric	EQ band 1 frequency (Hz, log)
`.eq.b1.gain`	numeric	EQ band 1 gain (dB)
`.eq.b1.q`	numeric	EQ band 1 Q (log)
`.comp.thresh`	numeric	Compressor threshold (dB)
`.comp.ratio`	numeric	Compressor ratio (log)
`.comp.attack`	numeric	Compressor attack (ms, log)
`.comp.release`	numeric	Compressor release (ms, log)
`.comp.gain`	numeric	Compressor makeup gain (dB)
`.geq.peak.N`	numeric	GEQ band N gain (dB)

4. Commands

4.1 Query — '?'

Read one or more key values by prefixing the key path with `?`.

```
?i.0.mix
i.0.mix=-10

?i.0.name
i.0.name=Kick

?i.0.mix,i.1.mix
i.0.mix=-10
i.1.mix=-5
```

4.2 Key Spec — '??'

Query the metadata (type, range, default, unit, description) for a key as JSON by prefixing with `??`.

```
??i.0.mix
??i.0.mix {"type":"double","min":-200,"max":10,"default":-1000,
           "unit":"dB","desc":"Input 1 Fader"}

??i.0.name
??i.0.name {"type":"string","default":"","desc":"Input 1 Name"}
```

4.3 Set — 'key=value'

Set a parameter to an absolute value using the `=` operator.

```
i.0.mix=-10
i.0.mute=1
i.0.name=Kick Drum
```

Wildcards and comma-separated lists work with set, applying the same value to every matched key:

```
i.*.mute=0                # unmute all inputs
i.0.mix,i.1.mix=-10       # set inputs 1 and 2 to -10 dB
%i.*.mix=0.75             # set all input faders to 75%
**.solo=0                 # clear all solos
```

String values preserve everything after the `=`, including spaces. Literal newlines in string values are encoded as `<NL>`.

4.4 Increment / Decrement — ‘+=’ / ‘-=’

Adjust a numeric parameter by a delta.

```
i.0.mix+=1.5
i.0.mix-=3
```

Wildcards and lists apply the same delta to every matched key:

```
i.*.mix+=3              # boost all input faders by 3 dB
%i.0.mix,%i.1.mix-=0.05 # nudge inputs 1 & 2 down by 5%
```

The result is clamped to each key’s valid range.

4.5 Toggle — ‘key!’

Toggle a boolean parameter (0 ↔️ 1). For parameters with a defined min/max range, toggles between min and max around the midpoint.

```
i.0.mute!
i.*.mute!                # toggle mute on all inputs
i.0.mute,i.1.mute!       # toggle mute on inputs 1 and 2
```

4.6 Subscribe — ‘+’

Subscribe to real-time value updates for matching keys. When subscribed, the server pushes `key=value` lines whenever the value changes. The current value is sent immediately upon subscribing.

```
+i.0.mix
i.0.mix=-10
```

Subsequent changes are pushed automatically:

```
i.0.mix=-8.5
i.0.mix=-7
```

4.7 Unsubscribe — ‘–‘

Remove a subscription. The pattern must exactly match the one used to subscribe.

```
-i.0.mix
```

4.8 Utility Commands

help

Prints a command summary as comment lines (`#`).

noop

Does nothing. Resets the idle timeout timer. No response is sent.

quit

Cleanly closes the connection. `exit` and `bye` are aliases.

5. Normalised Values

Prefix a key with `%` to work in a 0–1 normalised range instead of native units. This is useful for fader control surfaces and generic UIs that don't need to know the underlying parameter ranges or scaling.

The normalisation is scale-aware: logarithmic parameters (frequencies, attack/release times) are mapped through their log curve, and mix-scaled faders use the mixer's fader law.

```
?%i.0.mix
%i.0.mix=0.75

%i.0.mix=0.5

%i.0.mix+=0.05
%i.0.mix-=0.1
```

Normalised mode works with queries, sets, increments, decrements, and subscriptions:

```
+%i.0.mix
%i.0.mix=0.75
```

6. Wildcards

Wildcards expand to all matching keys in the keystore.

Pattern	Meaning
`*`	Matches a single dot-separated component
`**`	Matches one or more components (crosses dots)

Examples:

```
?i.0.*              # all parameters of input 1
?i.*.mix            # fader level of all inputs
?**.mute            # every mute on the mixer
?i.0.eq.b*.freq     # all EQ band frequencies on input 1
+i.*.mix            # subscribe to all input fader changes
```

Wildcards work with all commands that accept a key: `?`, `??`, `=`, `+=`, `-=`, `!`, `+`, `-`.

7. Comma-Separated Keys

Multiple keys (or patterns) can be combined in a single command with commas:

```
?i.0.mix,i.1.mix,i.2.mix
+%i.0.mix,%i.1.mix
```

8. Input Sanitisation

- Lines beginning with `#` are treated as comments and ignored.
- Tab characters are converted to spaces.
- Control characters below ASCII 32 (except tab and newline) are stripped.
- Leading spaces are trimmed.
- The `<NL>` escape sequence is converted to a literal newline on input and used to represent newlines in output.

9. Output Format

Response Format	Meaning
`key=value`	Native value (from query or subscription)
`%key=value`	Normalised 0–1 value
`??key {...}`	Key spec as JSON
`# text`	Comment (e.g. from help)

Numeric values are formatted with up to 10 significant digits, with trailing zeros removed.

10. Example Session

The following annotated session demonstrates a typical interaction with a VSKP-enabled mixer:

```
$ nc mixer.local 1703

# Query master fader
?m.mix
m.mix=0

# Set input 1 fader to -10 dB
i.0.mix=-10

# Read it back normalized
?%i.0.mix
%i.0.mix=0.7380964478

# Subscribe to all input mutes
+i.*.mute
i.0.mute=0
i.1.mute=0
i.2.mute=0
i.3.mute=0

# ...mute changes will be pushed as they happen...

# Toggle input 1 mute
i.0.mute!

# Bump input 2 fader up by 1 dB
i.1.mix+=1

# Get key metadata
??i.0.comp.thresh
??i.0.comp.thresh {"type":"double","min":-80,"max":0,
                   "default":-20,"unit":"dB",
                   "desc":"Input 1 Comp Threshold"}

# Set timeout and disconnect
tcptimeout=60
quit
```