teletype - manual

Contents

Introduction

Teletype is a dynamic, musical event triggering platform.

- Teletype Studies[1] - guided series of tutorials
- PDF command reference chart[2] — PDF scene recall sheet[3] — Default scenes[4]
- Current version: *5.0.0* — Firmware update procedure[5]

[1] https://monome.org/docs/modular/teletype/studies-1
[2] https://monome.org/docs/teletype/TT_commands_3.0.pdf
[3] https://monome.org/docs/teletype/TT_scene_RECALL_sheet.pdf
[4] http://monome.org/docs/teletype/scenes-10/
[5] https://monome.org/docs/modular/update/

Updates

v5.0.0

- **FIX**: fix off-by-one error in `P.ROT` understanding of pattern length
- **FIX**: fix `CROW.Q3` calls `ii.self.query2` instead of `ii.self.query3`
- **FIX**: cache currently-running commands to avoid corruption during SCENE ops.
- **FIX**: delay when opening docs
- **FIX**: `PROB 100` would execute only 99.01% of the time.
- **FIX**: some `G.FDR` configurations caused incorrect rendering in grid visualizer
- **FIX**: fix `EX.LP` not returning correct values
- **FIX**: fix `QT.B` handling of negative voltage input
- **IMP**: scene load/save code refactor, add scene load/save tests
- **IMP**: fader ops now support up to four faderbanks
- **NEW**: new Disting EX ops: dual algorithms, `EX.M.N#`, `EX.M.NO#`, `EX.M.CC#`
- **FIX**: reset M timer when changing metro rate
- **NEW**: drum ops: `DR.P`, `DR.V`, `DR.TR`
- **NEW**: I2C2MIDI[6] ops
- **FIX**: fix BPM rounding error
- **FIX**: support all line ending types for USB load
- **FIX**: fix `STATE` not accounting for `DEVICE.FLIP`
- **FIX**: fix MIDI IN ops channel number being off by 1
- **FIX**: improve `TR.P` accuracy
- **FIX**: fix `KILL` not stopping TR pulses in progress
- **NEW**: new op: `SCALE0` / `SCL0`
- **NEW**: new ops: `$F`, `$F1`, `$F2`, `$L`, `$L1`, `$L2`, `$S`, `$S1`, `$S2`, `I1`, `I2`, `FR`
- **NEW**: new op: `CV.GET`
- **NEW**: basic menu for reading/writing scenes when a USB stick is inserted
- **NEW**: new ops: `CV.CAL` and `CV.CAL.RESET` to calibrate CV outputs
- **FIX**: N.CS scales 7 & 8 were incorrectly swapped; make them consistent with N.S and docs
- **FIX**: libavr32 update: support CDC grid size detection (e.g. zero), increase HID message buffer
- **NEW**: new Disting EX ops: `EX.CH`, `EX.#`, `EX.N#`, `EX.NO#`
- **NEW**: apply VCV Rack compatibility patches, so branches off main can be used in both hardware and software
- **FIX**: update Disting EX looper ops to work with Disting EX firmware 1.23+
- **NEW**: new dual W/ ops: `W/.SEL`, `W/S.POLY`, `W/S.POLY.RESET`, `W/1`, `W/2`
- **NEW**: split cheatsheets into separate PDFs for core ops and i2c

[6] https://github.com/attowatt/i2c2midi

v4.0.0

- **FIX**: `LAST SCRIPT` in live mode gives time since init script was run
- **FIX**: negative pattern values are properly read from USB
- **FIX**: delay when navigating to sections in docs
- **NEW**: generic i2c ops: `IIA`, `IIS..`, `IIQ..`, `IIB..`
- **NEW**: exponential delay operator `DEL.G`
- **NEW**: binary and hex format for numbers: `B...`, `X...`
- **NEW**: Disting EX ops
- **FIX**: `LAST n` is broken for script 1
- **NEW**: bitmasked delay and quantize: `DEL.B..`, `QT.B..`, `QT.BX..`
- **NEW**: scale and chord quantize: `QT.S..`, `QT.CS..`
- **NEW**: bit toggle OP: `BTOG..`
- **NEW**: volts to semitones helper OP: `VN..`
- **IMP**: DELAY_SIZE increased to 64 from 16
- **FIX**: scale degree arguments 1-indexed: `N.S`, `N.CS`
- **NEW**: Just Friends 4.0 OPs and dual JF OPs
- **NEW**: binary scale ops `N.B` and `N.BX`
- **NEW**: reverse binary for numbers: `R...`
- **NEW**: reverse binary OP: `BREV`
- **NEW**: `ES.CV` read earthsea CV values
- **NEW**: added setter for R, sets R.MIN and R.MAX to same value, allowing R to be used as variable
- **NEW**: v/oct to hz/v conversion op: `HZ`
- **FIX**: W/2.0 ops added
- **NEW**: W/2.0 ops documentation
- **NEW**: ><, <>, >=< and <=> OPs, checks if value is within or outside of range
- **IMP**: new powerful Q OPs
- **IMP**: Improved line editing movement (forward/backward by word skips intervening space).
- **NEW**: Delete to end of word command `alt-d` added.
- **NEW**: new multi-logic OPs `AND3`, `AND4`, `OR3` and `OR4` with aliases &&&, &&&&, ||| and ||||
- **NEW**: ops to control live mode: `LIVE.OFF`, `LIVE.VARS`, `LIVE.GRID`, `LIVE.DASH`, `PRINT`
- **NEW**: `SCENE.P` OP: load another scene but keep current pattern state
- **NEW**: alias: `EV` for `EVERY`
- **NEW**: live mode dashboard
- **NEW**: ops to control live mode: `LIVE.OFF`, `LIVE.VARS`, `LIVE.GRID`, `LIVE.DASH`, `PRINT`
- **FIX**: `PN.ROT` parameters are swapped
- **FIX**: better rendering for fine grid faders
- **FIX**: logical operators should treat all non zero values as `true`, not just positive values
- **NEW**: crow ops
- **NEW**: `TI.PRM.CALIB` alias added (was already in the docs)
- **FIX**: SCENE would crash if parameter was out of bounds

v3.2.0

- **FIX**: improve DAC latency when using CV ops
- **NEW**: call metro / init with `SCRIPT 9` / `SCRIPT 10`
- **NEW**: forward (C-f or C-s) and reverse (C-r) search in help mode
- **NEW**: new ops: `LROT` (alias <<<), `RROT` (alias >>>)
- **NEW**: `LSH` and `RSH` shift the opposite direction when passed a negative shift amount
- **NEW**: new op: `SGN` (sign of argument)
- **NEW**: new kria remote op: `KR.DUR`
- **NEW**: new op: `NR` (binary math pattern generator)
- **NEW**: new ops: `N.S`, `N.C`, `N.CS` (use western scales and chords to get values from `N` table)
- **NEW**: new ops: `FADER.SCALE`, `FADER.CAL.MIN`, `FADER.CAL.MAX`, `FADER.CAL.RESET` for scaling 16n Faderbank values (aliases `FB.S`, `FB.C.MIN`, `FB.C.MAX`, `FB.C.R`)
- **NEW**: new Tracker mode keybinding `alt-[ ]` semitone up, down
- **NEW**: new Tracker mode keybinding `ctrl-[ ]` fifth up, down
- **NEW**: new Tracker mode keybinding `shift-[ ]` octave up, down
- **NEW**: new Tracker mode keybinding `alt-<0-9> <0-9>` semitones up (0=10, 1=11)
- **NEW**: new Tracker mode keybinding `shift-alt-<0-9> <0-9>` semitones down (0=10, 1=11)
- **FIX**: dim M in edit mode when metro inactive
- **NEW**: new pattern ops: `P.SHUF`, `PN.SHUF`, `P.REV`, `PN.REV`, `P.ROT`, `PN.ROT`
- **NEW**: new pattern mods: `P.MAP:`, `PN.MAP x:`

Version 3.1

New operators

`DEVICE.FLIP` - change how screen is displayed and how I/O are numbered to let you mount the module upside down

`DEL.X`, `DEL.R` - repeat an action multiple times, separated by a delay

J & K local script variables

`SEED`, `R.SEED`, `TOSS.SEED`, `DRUNK.SEED`, `P.SEED`, `PROB.SEED` - get/set seed for different random ops

`SCENE.G` - load another scene but keep the current grid configuration

`SCRIPT.POL` / `$.POL` - get / set script polarity. 1 to fire on rising edges as usual, 2 for falling edges, 3 for both. indicated on live mode w/ mutes icon.

New Ansible ops

`ANS.G` / `ANS.G.P` - simulate ansible receiving a grid key press

`ANS.A` - simulate ansible receiving an arc encoder turn

`ANS.G.LED` / `ANS.A.LED` - read LED brightness of ansible grid / arc

New Kria ops

`KR.CUE` - get / set the cued Kria pattern

`KR.PG` - switch to Kria parameter page

Changes

DELAY_SIZE increased to 16 from 8

Bug fixes

some keyboards losing keystrokes[7]

metro rate not updated after `INIT.SCENE`[8]

Version 3.0

Major new features

Grid Integration

Grid integration allows you to use grid to visualize, control and execute teletype scripts. You can create your own UIs using grid ops, or control Teletype directly with the Grid Control mode. Built in Grid Visualizer allows designing and using grid scenes without a grid. For more information and examples of grid scenes please see the Grid Studies[9].

Improved script editing

You can now select multiple lines when editing scripts by holding `shift`. You can move the current selection up and down with `alt-<up>` and `alt-<down>`. You can copy/cut/paste a multiline selection as well. To delete selected lines without copying into the clipboard use `alt-<delete>`.

Three level undo is also now available with `ctrl-z` shortcut.

Support for the Orthogonal Devices ER-301 Sound Computer over i2c

You now can connect up to three ER-301s via i2c and address up to 100 virtual CV channels and 100 virtual TR channels per ER-301. (The outputs range 1-100, 101-200, and

[7] https://github.com/monome/teletype/issues/156
[8] https://github.com/monome/teletype/issues/174
[9] https://github.com/scanner-darkly/teletype/wiki/GRID-INTEGRATION

201-300 respectively.) To function, this requires a slight mod to current in-market ER-301s and a specialized i2c cable that reorders two of the pins. Find more information on the Orthogonal Devices ER-301 Wiki Teletype Integration Page[10].

Support for the 16n Faderbank via i2c

The 16n Faderbank is an open-source sixteen fader controller with support for USB MIDI, standard MIDI, and i2c communication with the Teletype. It operates just like an IN or PARAM (or the TXi for that matter) in that you read values from the device. You use the operator FADER (or the alias FB) and the number of the slider you wish to poll (1-16). Know that longer cables may require that you use a powered bus board even if you only have one device on your Teletype's i2c bus. (You will know that you have a problem if your Teletype randomly hangs on reads.)

Support for the SSSR Labs SM010 Matrixarchate via i2c

The SSSR Labs SM010 Matrixarchate is a 16x8 IO Sequenceable Matrix Signal Router. Teletype integration allows you to switch programs and control connections. For a complete list of available ops refer to the manual. Information on how to connect the module can be found in the SM010 manual[11].

Support for W/ via i2c

Support for controlling Whimsical Raps W/ module via i2c. See the respective section for a complete list of available ops and refer to https://www.whimsicalraps.com/pages/w-type for more details.

New operators

`? x y z` is a ternary "if" operator, it will select between y and z based on the condition x.

New pattern ops

`P.MIN` `PN.MIN` `P.MAX` `PN.MAX` return the position for the first smallest/largest value in a pattern between the `START` and `END` points.

`P.RND` / `PN.RND` return a randomly selected value in a pattern between the `START` and `END` points.

`P.+` / `PN.+` / `P.-` / `PN.-` increment/decrement a pattern value by the specified amount.

`P.+W` / `PN.+W` / `P.-W` / `PN.-W` same as above and wrap to the specified range.

[10] http://wiki.orthogonaldevices.com/index.php/ER-301/Teletype_Integration
[11] https://www.sssrlabs.com/store/sm010/

New Telex ops

`TO.CV.CALIB` allows you to lock-in an offset across power cycles to calibrate your TELEX CV output (`TO.CV.RESET` removes the calibration).

`TO.ENV` now accepts gate values (1/0) to trigger the attack and decay.

New Kria ops

`KR.CV x` get the current CV value for channel x

`KR.MUTE x` `KR.MUTE x y` get/set mute state for channel x

`KR.TMUTE x` toggle mute state for channel x

`KR.CLK x` advance the clock for channel x

Ops for ER-301, 16n Faderbank, SM010, W/

Too many to list, please refer to their respective sections.

New aliases

`$` for `SCRIPT`

`RND` / `RRND` `RAND` / `RRAND`

`WRP` for `WRAP`

`SCL` for `SCALE`

New keybindings

Hold `shift` while making line selection in script editing to select multiple lines. Use `alt-<up>` and `alt-<down>` to move selected lines up and down. Copy/cut/paste shortcuts work with multiline selection as well. To delete selected lines without copying into the clipboard use `alt-<delete>`.

While editing a line you can now use `ctrl-<left>` / `ctrl-<right>` to move by words.

`ctrl-z` provides three level undo in script editing.

Additional `Alt-H` shortcut is available to view the Help screen.

`Alt-G` in Live mode will turn on the Grid Visualizer, which has its own shortcuts. Refer to the **Keys** section for a complete list.

The keybindings to insert a scaled knob value in the Tracker mode were changed from `ctrl` to `ctrl-alt` and from `shift` to `ctrl-shift`.

Bug fixes

i2c initialization delayed to account for ER-301 bootup

last screen saved to flash

knob jitter when loading/saving scenes reduced

duplicate commands not added to history[12]

`SCALE` precision improved

`PARAM` set properly when used in the init script

`PARAM` and `IN` won't reset to 0 after `INIT.DATA`

`PN.HERE`, `P.POP`, `PN.POP` will update the tracker screen[13]

`P.RM` was 1-based, now 0-based[14]

`P.RM` / `PN.RM` will not change pattern length if deleting outside of length range[15]

`JI` op fixed[16]

`TIME` and `LAST` are now 1ms accurate[17]

`RAND` / `RRAND` will properly work with large range values[18]

`L .. 32767` won't freeze[19]

New behavior

Previously, when pasting the clipboard while in script editing the pasted line would replace the current line. It will now instead push the current line down. This might result in some lines being pushed beyond the script limits - if this happens, use `ctrl-z` to undo the change, delete some lines and then paste again.

`I` would previously get initialized to 0 when executing a script. If you called a script from another script's loop this meant you had to use a variable to pass the loop's current `I` value to the called script. This is not needed anymore - when a script is called from another script its `I` value will be set to the current `I` value of the calling script.

Version 2.2

Teletype version 2.2 introduces Chaos and Bitwise operators, Live mode view of variables, INIT operator, ability to calibrate CV In and Param knob and set Min/Max scale

[12] https://github.com/monome/teletype/issues/99
[13] https://github.com/monome/teletype/issues/151
[14] https://github.com/monome/teletype/issues/149
[15] https://github.com/monome/teletype/issues/150
[16] https://llllllll.co/t/teletype-the-ji-op/10553
[17] https://github.com/monome/teletype/issues/144
[18] https://github.com/monome/teletype/issues/143
[19] https://github.com/monome/teletype/issues/148

values for both, a screensaver, Random Number Generator, and a number of fixes and improvements.

Major new features

Chaos Operators

The CHAOS operator provides a new source of uncertainty to the Teletype via chaotic yet deterministic systems. This operator relies on various chaotic maps for the creation of randomized musical events. Chaotic maps are conducive to creating music because fractals contain a symmetry of repetition that diverges just enough to create beautiful visual structures that at times also apply to audio. In mathematics a map is considered an evolution function that uses polynomials to drive iterative procedures. The output from these functions can be assigned to control voltages. This works because chaotic maps tend to repeat with slight variations offering useful oscillations between uncertainty and predictability.

Bitwise Operators

Bitwise operators have been added to compliment the logic functions and offer the ability to maximize the use of variables available on the Teletype.

Typically, when a variable is assigned a value it fully occupies that variable space; should you want to set another you'll have to use the next available variable. In conditions where a state of on, off, or a bitwise mathematical operation can provide the data required, the inclusion of these operators give users far more choices. Each variable normally contains 16 bits and Bitwise allows you to BSET, BGET, and BCLR a value from a particular bit location among its 16 positions, thus supplying 16 potential flags in the same variable space.

INIT

The new op family INIT features operator syntax for clearing various states from the unforgiving INIT with no parameters that clears ALL state data (be careful as there is no undo) to the ability to clear CV, variable data, patterns, scenes, scripts, time, ranges, and triggers.

Live Mode Variable Display

This helps the user to quickly check and monitor variables across the Teletype. Instead of single command line parameter checks the user is now able to simply press the ~ key (Tilde) and have a persistent display of eight system variables.

Screensaver

Screen saver engages after 90 minutes of inactivity

New Operators

- `IN.SCALE min max` sets the min/max values of the CV Input jack
- `PARAM.SCALE min max` set the min/max scale of the Parameter Knob
- `IN.CAL.MIN` sets the zero point when calibrating the CV Input jack
- `IN.CAL.MAX` sets the max point (16383) when calibrating the CV Input jack
- `PARAM.CAL.MIN` sets the zero point when calibrating the Parameter Kob
- `PARAM.CAL.MAX` sets the max point (16383) when calibrating the Parameter Kob
- `R` generate a random number
- `R.MIN` set the low end of the random number generator
- `R.MAX` set the upper end of the random number generator

Fixes

- Multiply now saturates at limits (-32768 / 32767) while previous behavior returned 0 at overflow
- Entered values now saturate at Int16 limits which are -32768 / 32767
- Reduced flash memory consumption by not storing TEMP script
- `I` now carries across `DEL` commands
- Corrected functionality of `JI` (Just Intonation) op for 1V/Oct tuning
- Reduced latency of `IN` op

Improvements

- Profiling code (optional developer feature)
- Screen now redraws only lines that have changed

Version 2.1

Teletype version 2.1 introduces new operators that mature the syntax and capability of the Teletype, as well as several bug fixes and enhancement features.

Major new features

Tracker Data Entry Improvements

Data entry in the tracker screen is now *buffered*, requiring an `ENTER` keystroke to commit changes, or `SHIFT-ENTER` to insert the value. All other navigation keystrokes will abandon data entry. The increment / decrement keystrokes (`]` and `[`), as well as the negate keystroke (`-`) function immediately if not in data entry mode, but modify the currently buffered value in edit mode (again, requiring a commit).

Turtle Operator

The Turtle operator allows 2-dimensional access to the patterns as portrayed out in Tracker mode. It uses new operators with the `@` prefix. You can `@MOVE X Y` the turtle

relative to its current position, or set its direction in degrees with @DIR and its speed with @SPEED and then execute a @STEP.

To access the value that the turtle operator points to, use @, which can also set the value with an argument.

The turtle can be constrained on the tracker grid by setting its fence with @FX1, @FY1, @FX2, and @FY2, or by using the shortcut operator @F x1 y1 x2 y2. When the turtle reaches the fence, its behaviour is governed by its *fence mode*, where the turtle can simply stop (@BUMP), wrap around to the other edge (@WRAP), or bounce off the fence and change direction (@BOUNCE). Each of these can be set to 1 to enable that mode.

Setting @SCRIPT N will cause script N to execute whenever the turtle crosses the boundary to another cell. This is different from simply calling @STEP; @SCRIPT N because the turtle is not guaranteed to change cells on every step if it is moving slowly enough.

Finally, the turtle can be displayed on the tracker screen with @SHOW 1, where it will indicate the current cell by pointing to it from the right side with the < symbol.

New Mods: EVERY, SKIP, and OTHER, plus SYNC

These mods allow rhythmic division of control flow. EVERY X: executes the post-command once per X at the Xth time the script is called. SKIP X: executes it every time but the Xth. OTHER: will execute when the previous EVERY/SKIP command did not.

Finally, SYNC X will set each EVERY and SKIP counter to X without modifying its divisor value. Using a negative number will set it to that number of steps before the step. Using SYNC -1 will cause each EVERY to execute on its next call, and each SKIP will not execute.

Script Line "Commenting"

Individual lines in scripts can now be disabled from execution by highlighting the line and pressing ALT-/. Disabled lines will appear dim. This status will persist through save/load from flash, but will not carry over to scenes saved to USB drive.

New Operators

W [condition]: is a new mod that operates as a while loop. The BREAK operator stops executing the current script BPM [bpm] returns the number of milliseconds per beat in a given BPM, great for setting M. LAST [script] returns the number of milliseconds since script was last called.

New Operator Behaviour

SCRIPT with no argument now returns the current script number. I is now local to its corresponding L statement. IF/ELSE is now local to its script.

New keybindings

`CTRL-1` through `CTRL-8` toggle the mute status for scripts 1 to 8 respectively. `CTRL-9` toggles the METRO script. `SHIFT-ENTER` now inserts a line in Scene Write mode.

Bug fixes

Temporal recursion now possible by fixing delay allocation issue, e.g.: DEL 250: SCRIPT SCRIPT `KILL` now clears `TR` outputs and stops METRO. `SCENE` will no longer execute from the INIT script on initial scene load. `AVG` and `Q.AVG` now round up from offsets of 0.5 and greater.

Breaking Changes

As `I` is now local to `L` loops, it is no longer usable across scripts or as a general-purpose variable. As `IF/ELSE` is now local to a script, scenes that relied on IF in one script and ELSE in another will be functionally broken.

Version 2.0

Teletype version 2.0 represents a large rewrite of the Teletype code base. There are many new language additions, some small breaking changes and a lot of under the hood enhancements.

Major new features

Sub commands

Several commands on one line, separated by semicolons.

e.g. `CV 1 N 60; TR.PULSE 1`

See the section on “Sub commands” for more information.

Aliases

For example, use `TR.P 1` instead of `TR.PULSE 1`, and use `+ 1 1`, instead of `ADD 1 1`.

See the section on “Aliases” for more information.

PN versions of every P OP

There are now `PN` versions of every `P OP`. For example, instead of:

```
P.I 0
P.START 0
P.I 1
```

```
P.START 10
```

You can use:

```
PN.START 0 0
PN.START 1 10
```

TELEXi and TELEXo OPs

Lots of OPs have been added for interacting with the wonderful TELEXi input expander and TELEXo output expander. See their respective sections in the documentation for more information.

New keybindings

The function keys can now directly trigger a script.

The `<tab>` key is now used to cycle between live, edit and pattern modes, and there are now easy access keys to directly jump to a mode.

Many new text editing keyboard shortcuts have been added.

See the “Modes” documentation for a listing of all the keybindings.

USB memory stick support

You can now save you scenes to USB memory stick at any time, and not just at boot up. Just insert a USB memory stick to start the save and load process. Your edit scene should not be effected.

It should also be significantly more reliable with a wider ranger of memory sticks.

WARNING: Please backup the contents of your USB stick before inserting it. Particularly with a freshly flashed Teletype as you will end up overwriting all the saved scenes with blank ones.

Other additions

- Limited script recursion now allowed (max recursion depth is 8) including self recursion.
- Metro scripts limited to 25ms, but new `M!` op to set it as low as 2ms (at your own risk), see “Metronome” OP section for more.

Breaking changes

- **Removed the need for the `II` OP.**

  For example, `II MP.PRESET 1` will become just `MP.PRESET 1`.

- **Merge `MUTE` and `UNMUTE` OPs to `MUTE x` / `MUTE x y`.**

  See the documentation for `MUTE` for more information.

- **Remove unused Meadowphysics OPs.**

  Removed: `MP.SYNC`, `MP.MUTE`, `MP.UNMUTE`, `MP.FREEZE`, `MP.UNFREEZE`.

- **Rename Ansible Meadowphysics OPs to start with `ME`.**

  This was done to avoid conflicts with the Meadowphysics OPs.

WARNING: If you restore your scripts from a USB memory stick, please manually fix any changes first. Alternatively, incorrect commands (due to the above changes) will be skipped when imported, please re-add them.

Known issues

Visual glitches

The cause of these is well understood, and they are essentially harmless. Changing modes with the `<tab>` key will force the screen to redraw. A fix is coming in version 2.1.

Quickstart

Panel

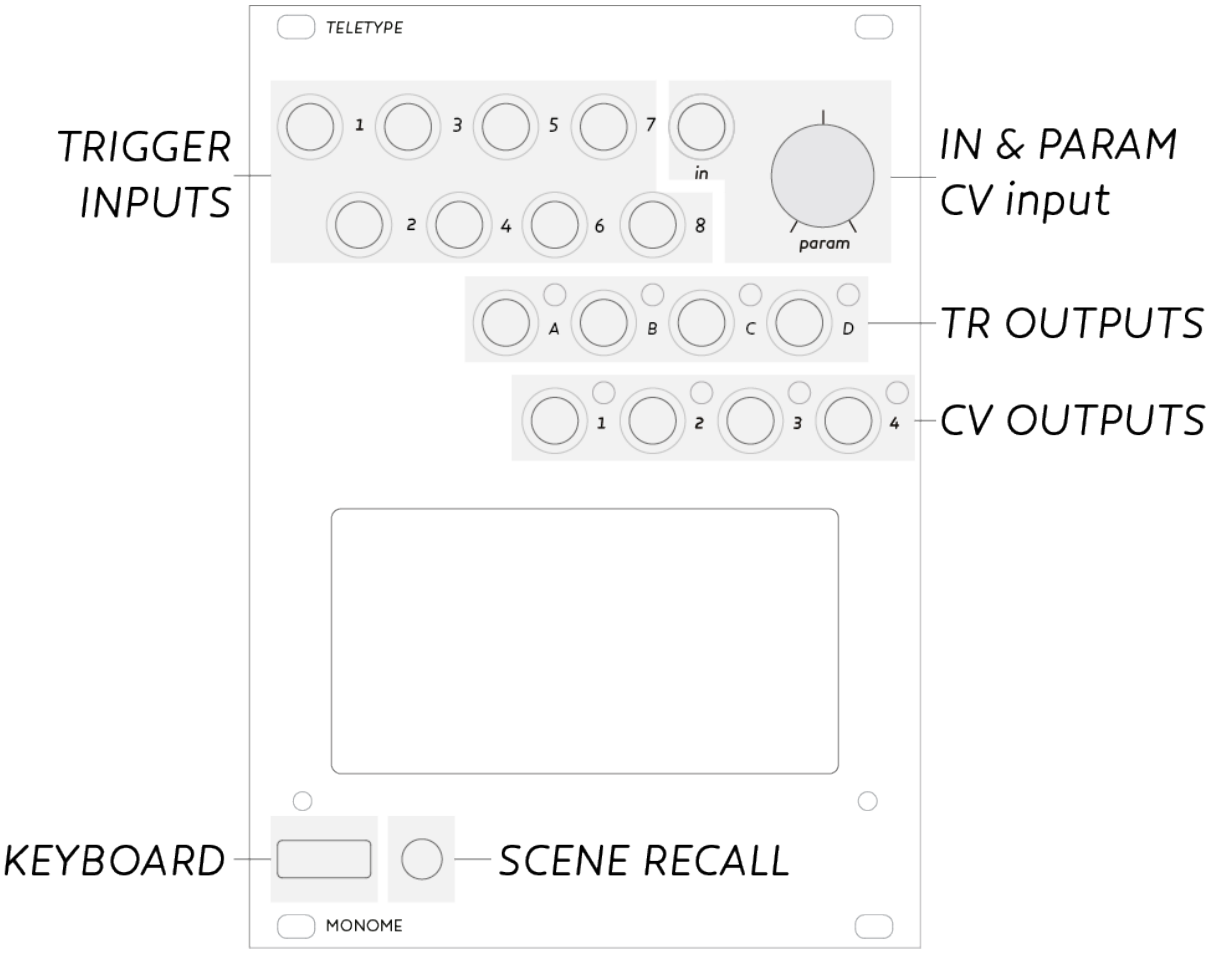

Figure 1: Panel Overlay

The keyboard is attached to the front panel, for typing commands. The commands can be executed immediately in *LIVE mode* or assigned to one of the eight trigger inputs in *EDIT mode*. The knob and in jack can be used to set and replace values.

LIVE mode

Teletype starts up in LIVE mode. You'll see a friendly **>** prompt, where commands are entered. The command:

```
TR.TOG A
```

will toggle trigger A after pressing enter. Consider:

```
CV 1 V 5
CV 2 N 7
CV 1 0
```

Here the first command sets CV 1 to 5 volts. The second command sets CV 2 to note 7 (which is 7 semitones up). The last command sets CV 1 back to 0.

Data flows from right to left, so it becomes possible to do this:

```
CV 1 N RAND 12
```

Here a random note between 0 and 12 is set to CV 1.

We can change the behavior of a command with a *PRE* such as DEL:

```
DEL 500 : TR.TOG A
```

`TR.TOG A` will be delayed by 500ms upon execution.

A helpful display line appears above the command line in dim font. Here any entered commands will return their numerical value if they have one.

SCRIPTS, or several lines of commands, can be assigned to trigger inputs. This is when things get musically interesting. To edit each script, we shift into EDIT mode.

LIVE mode icons

Four small icons are displayed in LIVE mode to give some important feedback about the state of Teletype. These icons will be brightly lit when the above is true, else will remain dim. They are, from left to right:

- Slew: CV outputs are currently slewing to a new destination.
- Delay: Commands are in the delay queue to be executed in the future.
- Stack: Commands are presently on the stack waiting for execution.
- Metro: Metro is currently active and the Metro script is not empty.

EDIT mode

Toggle between EDIT and LIVE modes by pushing **TAB**.

The prompt now indicates the script you're currently editing:

- 1-8 indicates the script associated with corresponding trigger
- `M` is for the internal metronome
- `I` is the init script, which is executed upon scene recall

Script 1 will be executed when trigger input 1 (top left jack on the panel) receives a low-to-high voltage transition (trigger, or front edge of a gate). Consider the following as script 1:

1:

```
TR.TOG A
```

Now when input 1 receives a trigger, `TR.TOG A` is executed, which toggles the state of output trigger A.

Scripts can have multiple lines:

1:

```
TR.TOG A
CV 1 V RAND 4
```

Now each time input 1 receives a trigger, CV 1 is set to a random volt between 0 and 4, in addition to output trigger A being toggled.

Metronome

The `M` script is driven by an internal metronome, so no external trigger is required. By default the metronome interval is 1000ms. You can change this readily (for example, in LIVE mode):

```
M 500
```

The metronome interval is now 500ms. You can disable/enable the metronome entirely with `M.ACT`:

```
M.ACT 0
```

Now the metronome is off, and the `M` script will not be executed. Set `M.ACT` to 1 to re-enable.

Patterns

Patterns facilitate musical data manipulation– lists of numbers that can be used as sequences, chord sets, rhythms, or whatever you choose. Pattern memory consists four banks of 64 steps. Functions are provided for a variety of pattern creation, transformation, and playback. The most basic method of creating a pattern is by directly adding numbers to the sequence:

```
P.PUSH 5
P.PUSH 11
P.PUSH 9
P.PUSH 3
```

`P.PUSH` adds the provided value to the end of the list– patterns keep track of their length, which can be read or modified with `P.L`. Now the pattern length is 4, and the list looks something like:

```
5, 11, 9, 3
```

Patterns also have an index `P.I`, which could be considered a playhead. `P.NEXT` will advance the index by one, and return the value stored at the new index. If the playhead

hits the end of the list, it will either wrap to the beginning (if P.WRAP is set to 1, which it is by default) or simply continue reading at the final position.

So, this script on input 1 would work well:

1:

```
CV 1 N P.NEXT
```

Each time input 1 is triggered, the pattern moves forward one then CV 1 is set to the note value of the pattern at the new index. This is a basic looped sequence. We could add further control on script 2:

2:

```
P.I 0
```

Since P.I is the playhead, trigger input 2 will reset the playhead back to zero. It won't change the CV, as that only happens when script 1 is triggered.

We can change a value within the pattern directly:

```
P 0 12
```

This changes index 0 to 12 (it was previously 5), so now we have *12, 11, 9, 3*.

We've been working with pattern 0 up to this point. There are four pattern banks, and we can switch banks this way:

```
P.N 1
```

Now we're on pattern bank 1. P.NEXT, P.PUSH, P, (and several more commands) all reference the current pattern bank. Each pattern maintains its own play index, wrap parameter, length, etc.

We can directly access and change *any* pattern value with the command PN:

```
PN 3 0 22
```

Here the first argument (3) is the *bank*, second (0) is the *index*, and last is the new value (22). You could do this by doing P.N 3 then P 0 22 but there are cases where a direct read/write is needed in your patch.

Check the *Command Set* section below for more pattern commands.

Patterns are stored in flash with each scene!

TRACKER mode

Editing patterns with scripts or from the command line isn't always ergonomic. When you'd like to visually edit patterns, TRACKER mode is the way.

The TAB key cycles between LIVE, EDIT and TRACKER mode. You can also get directly to TRACKER mode by pressing the NUM LOCK key. TRACKER mode is the one with 4 columns of numbers on the Teletype screen.

The current pattern memory is displayed in these columns. Use the arrow keys to navigate. Holding ALT will jump by pages.

The edit position is indicated by the brightest number. Very dim numbers indicate they are outside the pattern length.

Use the square bracket keys [and] to decrease/increase the values. Backspace sets the value to 0. Entering numbers will overwrite a new value. You can cut/copy/paste with ALT-X-C-V.

Check the *Keys* section for a complete list of tracker shortcuts.

Scenes

A *SCENE* is a complete set of scripts and patterns. Stored in flash, scenes can be saved between sessions. Many scenes ship as examples. On startup, the last used scene is loaded by Teletype.

Access the SCENE menu using ESCAPE. The bracket keys ([and]) navigate between the scenes. Use the up/down arrow keys to read the scene *text*. This text will/should describe what the scene does generally along with input/output functions. ENTER will load the selected scene, or ESCAPE to abort.

To save a scene, hold ALT while pushing ESCAPE. Use the brackets to select the destination save position. Edit the text section as usual– you can scroll down for many lines. The top line is the name of the scene. ALT-ENTER will save the scene to flash.

Keyboard-less Scene Recall

To facilitate performance without the need for the keyboard, scenes can be recalled directly from the module's front panel.

- Press the SCENE RECALL button next to the USB jack on the panel.
- Use the PARAM knob to highlight your desired preset.
- Hold the SCENE RECALL button for 1 second to load the selected scene.

Init Script

The *INIT* script (represented as I) is executed when a preset is recalled. This is a good place to set initial values of variables if needed, like metro time M or time enable TIME.ACT for example.

USB Backup

Teletype's scenes can be saved and loaded from a USB flash drive. When a flash drive is inserted, Teletype will recognize it and go into disk mode. First, all 32 scenes will be written to text files on the drive with names of the form tt##s.txt. For example, scene 5 will be saved to tt05s.txt. The screen will display WRITE..... as this is done.

Once complete, Teletype will attempt to read any files named `tt##.txt` and load them into memory. For example, a file named `tt13.txt` would be loaded as scene 13 on Teletype. The screen will display `READ.....` Once this process is complete, Teletype will return to LIVE mode and the drive can be safely removed.

For best results, use an FAT-formatted USB flash drive. If Teletype does not recognize a disk that is inserted within a few seconds, it may be best to try another.

An example of possible scenes to load, as well as the set of factory default scenes, can be found at the Teletype Codex[20].

Commands

Nomenclature

- SCRIPT – multiple *commands*
- COMMAND – a series (one line) of *words*
- WORD – a text string separated by a space: *value*, *operator*, *variable*, *mod*
- VALUE – a number
- OPERATOR – a function, may need value(s) as argument(s), may return value
- VARIABLE – named memory storage
- MOD – condition/rule that applies to rest of the *command*, e.g.: del, prob, if, s

Syntax

Teletype uses prefix notation. Evaluation happens from right to left.

The left value gets assignment (*set*). Here, temp variable X is assigned zero:

```
X 0
```

Temp variable Y is assigned to the value of X:

```
Y X
```

X is being *read* (*get* X), and this value is being used to *set* Y.

Instead of numbers or variables, we can use operators to perform more complex behavior:

```
X TOSS
```

TOSS returns a random state, either 0 or 1 on each call.

Some operators require several arguments:

```
X ADD 1 2
```

Here ADD needs two arguments, and gets 1 and 2. X is assigned the result of ADD, so X is now 3.

[20] https://github.com/monome-community/teletype-codex

If a value is returned at the end of a command, it is printed as a MESSAGE. This is visible in LIVE mode just above the command prompt. (In the examples below ignore the // comments).

```
8           // prints 8
X 4
X           // prints 4
ADD 8 32    // prints 40
```

Many parameters are indexed, such as CV and TR. This means that CV and TR have multiple values (in this case, each has four.) We pass an extra argument to specify which index we want to read or write.

```
CV 1 0
```

Here CV 1 is set to 0. You can leave off the 0 to print the value.

```
CV 1        // prints value of CV 1
```

Or, this works too:

```
X CV 1      // set X to current value of CV 1
```

Here is an example of using an operator RAND to set a random voltage:

```
CV 1 V RAND 4
```

First a random value between 0 and 3 is generated. The result is turned into a volt with a table lookup, and the final value is assigned to CV 1.

The order of the arguments is important, of course. Consider:

```
CV RRAND 1 4 0
```

RRAND uses two arguments, 1 and 4, returning a value between these two. This command, then, chooses a random CV output (1-4) to set to 0. This might seem confusing, so it's possible to clarify it by pulling it apart:

```
X RRAND 1 4
CV X 0
```

Here we use X as a temp step before setting the final CV.

With some practice it becomes easier to combine many functions into the same command.

Furthermore, you can use a semicolon to include multiple commands on the same line:

```
X RRAND 1 4; CV X 0
```

This is particularly useful in **INIT** scripts where you may want to initialize several values at once:

```
A 66; X 101; TR.TIME 1 20;
```

Continuing

Don’t forget to checkout the Teletype Studies[21] for an example-driven guide to the language.

[21] https://monome.org/docs/modular/teletype/studies-1

Keys

Global key bindings

These bindings work everywhere.

Key	Action
<tab>	change modes, live to edit to pattern and back
<esc>	preset read mode, or return to last mode
alt-<esc>	preset write mode
win-<esc>	clear delays, stack and slews
shift-alt-? / **alt-h**	help text, or return to last mode
<F1> to **<F8>**	run corresponding script
<F9>	run metro script
<F10>	run init script
alt-<F1> to **alt-<F8>**	edit corresponding script
alt-<F9>	edit metro script
alt-<F10>	edit init script
ctrl-<F1> to **ctrl-<F8>**	mute/unmute corresponding script
ctrl-<F9>	enable/disable metro script
<numpad-1> to **<numpad-8>**	run corresponding script
<num lock> / **<F11>**	jump to pattern mode
<print screen> / **<F12>**	jump to live mode

Text editing

These bindings work when entering text or code.

In most cases, the clipboard is shared between *live*, *edit* and the 2 *preset* modes.

Key	Action
<left> / **ctrl-b**	move cursor left
<right> / **ctrl-f**	move cursor right
ctrl-<left> / **alt-b**	move left by one word
ctrl-<right> / **alt-f**	move right by one word
<home> / **ctrl-a**	move to beginning of line

Key	Action
<end> / **ctrl-e**	move to end of line
<backspace> / **ctrl-h**	backwards delete one character
<delete> / **ctrl-d**	forwards delete one character
shift-<backspace> / **ctrl-u**	delete from cursor to beginning
shift-<delete> / **ctrl-k**	delete from cursor to end
alt-<backspace> / **ctrl-w**	delete from cursor to beginning of word
alt-d	delete from cursor to end of word
ctrl-x / **alt-x**	cut to clipboard
ctrl-c / **alt-c**	copy to clipboard
ctrl-v / **alt-v**	paste to clipboard

Live mode

Key	Action
<down> / **C-n**	history next
<up> / **C-p**	history previous
<enter>	execute command
~	toggle variables
[/ **]**	switch to edit mode
alt-g	toggle grid visualizer
shift-d	live dashboard
alt-<arrows>	move grid cursor
alt-shift-<arrows>	select grid area
alt-<space>	emulate grid press
alt-/	switch grid pages
alt-	toggle grid control view
alt-<prt sc>	insert grid x/y/w/h

In full grid visualizer mode pressing `alt` is not required.

Edit mode

In *edit* mode multiple lines can be selected and used with the clipboard.

Key	Action
<down> / **C-n**	line down
<up> / **C-p**	line up
[	previous script
]	next script
<enter>	enter command
shift-<enter>	insert command
alt-/	toggle line comment
shift-<up>	expand selection up
shift-<down>	expand selection down
alt-<delete>	delete selection
alt-<up>	move selection up
alt-<down>	move selection down
ctrl-z	undo (3 levels)

Tracker mode

The tracker mode clipboard is independent of text and code clipboard.

Key	Action
<down>	move down
alt-<down>	move a page down
<up>	move up
alt-<up>	move a page up
<left>	move left
alt-<left>	move to the very left
<right>	move right
alt-<right>	move to the very right
[	decrement by 1
]	increment by 1
alt-[	decrement by 1 semitone
alt-]	increment by 1 semitone
ctrl-[	decrement by 7 semitones
ctrl-]	increment by 7 semitones
shift-[	decrement by 12 semitones
shift-]	increment by 12 semitones
alt-<0-9>	increment by <0-9> semitones (0=10, 1=11)

Key	Action
shift-alt-<0-9>	decrement by <0-9> semitones (0=10, 1=11)
<backspace>	delete a digit
shift-<backspace>	delete an entry, shift numbers up
<enter>	commit edit (increase length if cursor in position after last entry)
shift-<enter>	commit edit, then duplicate entry and shift downwards (increase length as <enter>)
alt-x	cut value (n.b. ctrl-x not supported)
alt-c	copy value (n.b. ctrl-c not supported)
alt-v	paste value (n.b. ctrl-v not supported)
shift-alt-v	insert value
shift-l	set length to current position
alt-l	go to current length entry
shift-s	set start to current position
alt-s	go to start entry
shift-e	set end to current position
alt-e	go to end entry
-	negate value
<space>	toggle non-zero to zero, and zero to 1
0 to **9**	numeric entry
shift-2 (@)	toggle turtle display marker (<)
ctrl-alt	insert knob value scaled to 0..31
ctrl-shift	insert knob value scaled to 0..1023

Preset read mode

Key	Action
<down> / **C-n**	line down
<up> / **C-p**	line up
<left> / **[**	preset down
<right> / **]**	preset up
<enter>	load preset

Preset write mode

Key	Action
<down> / **C-n**	line down
<up> / **C-p**	line up
[	preset down
]	preset up
<enter>	enter text
shift-<enter>	insert text
alt-<enter>	save preset

Help mode

Key	Action
<down> / **C-n**	line down
<up> / **C-p**	line up
<left> / **[**	previous page
<right> / **]**	next page
C-f / **C-s**	search forward
C-r	search backward

OPs and MODs

Variables

General purpose temp vars: X, Y, Z, and T.

T typically used for time values, but can be used freely.

A-D are assigned 1-4 by default (as a convenience for TR labeling, but TR can be addressed with simply 1-4). All may be overwritten and used freely.

OP	OP *(set)*	*(aliases)*	Description
A	**A x**		get / set the variable A, default 1
B	**B x**		get / set the variable B, default 2
C	**C x**		get / set the variable C, default 3
D	**D x**		get / set the variable D, default 4
FLIP	**FLIP x**		returns the opposite of its previous state (0 or 1) on each read (also settable)
I	**I x**		get / set the variable I
J	**J x**		get / set the variable J
K	**K x**		get / set the variable K
O	**O x**		auto-increments *after* each access, can be set, starting value 0
O.INC	**O.INC x**		how much to increment O by on each invocation, default 1
O.MIN	**O.MIN x**		the lower bound for O, default 0
O.MAX	**O.MAX x**		the upper bound for O, default 63
O.WRAP	**O.WRAP x**		should O wrap when it reaches its bounds, default 1
T	**T x**		get / set the variable T, typically used for time, default 0
TIME	**TIME x**		timer value, counts up in ms., wraps after 32s, can be set
TIME.ACT	**TIME.ACT x**		enable or disable timer counting, default 1
LAST x			get value in milliseconds since last script run time
X	**X x**		get / set the variable X, default 0
Y	**Y x**		get / set the variable Y, default 0
Z	**Z x**		get / set the variable Z, default 0

I

- **I** / **I x**

Get / set the variable `I`. This variable is overwritten by L, but can be used freely outside an L loop. Each script gets its own `I` variable, so if you call a script from another script's loop you can still use and modify `I` without affecting the calling loop. In this scenario the script getting called will have its `I` value initialized with the calling loop's current `I` value.

J

- **J** / **J x**

get / set the variable J, each script gets its own J variable, so if you call a script from another script you can still use and modify J without affecting the calling script.

K

- **K** / **K x**

get / set the variable K, each script gets its own K variable, so if you call a script from another script you can still use and modify K without affecting the calling script.

O

- **O** / **O x**

Auto-increments by `O.INC` *after* each access. The initial value is `0`. The lower and upper bounds can be set by `O.MIN` (default `0`) and `O.MAX` (default 63). `O.WRAP` controls if the value wraps when it reaches a bound (default is 1).

Example:

```
O           => 0
O           => 1
X O
X           => 2
O.INC 2
O           => 3 (O increments after it's accessed)
O           => 5
O.INC -2
O 2
O           => 2
O           => 0
O           => 63
O           => 61
```

LAST

- **LAST x**

Gets the number of milliseconds since the given script was run, where `M` is script 9 and `I` is script 10. From the live mode, `LAST SCRIPT` gives the time elapsed since last run of I script.

For example, one-line tap tempo:

```
M LAST SCRIPT
```

Running this script twice will set the metronome to be the time between runs.

Hardware

The Teletype trigger inputs are numbered 1-8, the CV and trigger outputs 1-4. See the Ansible documentation for details of the Ansible output numbering when in Teletype mode.

OP	OP *(set)*	*(aliases)*	Description
CV x	**CV x y**		CV target value
CV.OFF x	**CV.OFF x y**		CV offset added to output
CV.SET x y			Set CV value, ignoring slew
CV.GET x			Get current CV value
CV.SLEW x	**CV.SLEW x y**		Get/set the CV slew time in ms
V x			converts a voltage to a value usable by the CV outputs (x between `0` and `10`)
VV x			converts a voltage to a value usable by the CV outputs (x between `0` and `1000`, `100` represents 1V)
IN			Get the value of IN jack (0-16383)
IN.SCALE min max			Set static scaling of the `IN` CV to between `min` and `max`.
PARAM		**PRM**	Get the value of PARAM knob (0-16383)
PARAM.SCALE min max			Set static scaling of the PARAM knob to between `min` and `max`.
TR x	**TR x y**		Set trigger output x to y (0-1)
TR.PULSE x		**TR.P**	Pulse trigger output x
TR.TIME x	**TR.TIME x y**		Set the pulse time of trigger x to y ms
TR.TOG x			Flip the state of trigger output x
TR.POL x	**TR.POL x y**		Set polarity of trigger output x to y (0-1)
MUTE x	**MUTE x y**		Disable trigger input x
STATE x			Read the current state of input x
LIVE.OFF		**LIVE.O**	Show the default live mode screen
LIVE.VARS		**LIVE.V**	Show variables in live mode
LIVE.GRID		**LIVE.G**	Show grid visualizer in live mode
LIVE.DASH x		**LIVE.D**	Show the dashboard with index x

OP	OP *(set)*	*(aliases)*	Description
PRINT x	**PRINT x y**	**PRT**	Print a value on a live mode dashboard or get the printed value

CV

- **CV x** / **CV x y**

Get the value of CV associated with output x, or set the CV output of x to y.

CV.OFF

- **CV.OFF x** / **CV.OFF x y**

Get the value of the offset added to the CV value at output x. The offset is added at the final stage. Set the value of the offset added to the CV value at output x to y.

CV.SET

- **CV.SET x y**

Set the CV value at output x bypassing any slew settings.

CV.GET

- **CV.GET x**

Get the current CV value at output x with slew and offset applied.

CV.SLEW

- **CV.SLEW x** / **CV.SLEW x y**

Get the slew time in ms associated with CV output x. Set the slew time associated with CV output x to y ms.

IN

- **IN**

Get the value of the IN jack. This returns a valuue in the range 0-16383.

PARAM

- **PARAM**
- *alias*: **PRM**

Get the value of the PARAM knob. This returns a valuue in the range 0-16383.

TR

- **TR x / TR x y**

Get the current state of trigger output x. Set the state of trigger output x to y (0-1).

TR.PULSE

- **TR.PULSE x**
- *alias*: **TR.P**

Pulse trigger output x.

TR.TIME

- **TR.TIME x / TR.TIME x y**

Get the pulse time of trigger output x. Set the pulse time of trigger output x to yms.

TR.TOG

- **TR.TOG x**

Flip the state of trigger output x.

TR.POL

- **TR.POL x / TR.POL x y**

Get the current polarity of trigger output x. Set the polarity of trigger output x to y (0-1). When TR.POL = 1, the pulse is 0 to 1 then back to 0. When TR.POL = 0, the inverse is true, 1 to 0 to 1.

MUTE

- **MUTE x / MUTE x y**

Mute the trigger input on x (1-8) when y is non-zero.

STATE

- **STATE x**

Read the current state of trigger input x (0=low, 1=high).

LIVE.DASH

- **LIVE.DASH x**
- *alias*: **LIVE.D**

This allows you to show custom text and print values on the live mode screen. To create a dashboard, simply edit the scene description. You can define multiple dashboards by separating them with ===, and you can select them by specifying the dashboard number as the op parameter.

You can also print up to 16 values using PRINT op. To create a placeholder for a value, place %## where you want the number to be, where ## is a value index between 1 and 16. Please note: if you define multiple placeholders for the same value, only the last one will be used, and the rest will be treated as plain text. By default, values are printed in decimal format, but you can also use hex, binary and reversed binary formats by using %X##, %B## and %R## placeholders respectively.

An example of a dashboard:

```
THIS IS A DASHBOARD

CURRENT METRO RATE IS: %1
```

You can use this dashboard by entering the above in a scene description, placing `LIVE.DASH 1` in the init script and placing `PRINT 1 M` in the metro script.

PRINT

- **PRINT x** / **PRINT x y**
- *alias*: **PRT**

This op allows you to display up to 16 values on a live mode dashboard and should be used in conjunction with `LIVE.DASH` op. See `LIVE.DASH` description for information on how to use it. You can also use this op to store up to 16 additional values.

Pitch

Mathematical calcuations and tables helpful for musical pitch.

OP	OP *(set)*	*(aliases)*	Description
`HZ x`			converts 1V/OCT value x to Hz/Volt value, useful for controlling non-euro synths like Korg MS-20
`JI x y`			just intonation helper, precision ratio divider normalised to 1V
`N x`			converts an equal temperament note number to a value usable by the CV outputs (x in the range -127 to 127)
`N.S r s d`			Note Scale operator, `r` is the root note (`0`-127), s is the scale (`0`-8) and d is the degree (1-7), returns a value from the N table.
`N.C r c d`			Note Chord operator, `r` is the root note (`0`-127), c is the chord (`0`-12) and d is the degree (`0`-3), returns a value from the N table.
`N.CS r s d c`			Note Chord Scale operator, `r` is the root note (`0`-127), s is the scale (`0`-8), d is the scale degree (1-7) and c is the chord component (`0`-3), returns a value from the N table.
`N.B d`	**`N.B r s`**		get degree d of scale/set scale root to `r`, scale to s, s is either bit mask (s >= 1) or scale preset (s < 1)
`N.BX i d`	**`N.BX i r s`**		multi-index version of N.B, scale at `i` (index) 0 is shared with N.B
`VN x`			converts 1V/OCT value x to an equal temperament note number
`QT.B x`			quantize 1V/OCT signal x to scale defined by `N.B`
`QT.BX i x`			quantize 1V/OCT signal x to scale defined by `N.BX` in scale index `i`
`QT.S x r s`			quantize 1V/OCT signal x to scale s (0-8, reference N.S scales) with root 1V/OCT pitch `r`

OP	OP *(set)*	*(aliases)*	Description
`QT.CS x r s d c`			quantize 1V/OCT signal x to chord c (1-7) from scale s (0-8, reference N.S scales) at degree d (1-7) with root 1V/OCT pitch r

N

- **`N x`**

The N OP converts an equal temperament note number to a value usable by the CV outputs.

Examples:

```
CV 1 N 60         => set CV 1 to middle C, i.e. 5V
CV 1 N RAND 24  => set CV 1 to a random note from the lowest 2 octaves
```

N.S

- **`N.S r s d`**

The `N.S` OP lets you retrieve N table values according to traditional western scales. s and d wrap to their ranges automatically and support negative indexing.

Scales - `0` = Major - 1 = Natural Minor - 2 = Harmonic Minor - 3 = Melodic Minor - 4 = Dorian - 5 = Phrygian - 6 = Lydian - 7 = Mixolydian - 8 = Locrian

N.C

- **`N.C r c d`**

The `N.C` OP lets you retrieve N table values according to traditional western chords. c and d wrap to their ranges automatically and support negative indexing.

Chords - `0` = Major 7th `{0, 4, 7, 11}` - 1 = Minor 7th `{0, 3, 7, 10}` - 2 = Dominant 7th `{0, 4, 7, 10}` - 3 = Diminished 7th `{0, 3, 6, 9}` - 4 = Augmented 7th `{0, 4, 8, 10}` - 5 = Dominant 7b5 `{0, 4, 6, 10}` - 6 = Minor 7b5 `{0, 3, 6, 10}` - 7 = Major 7#5 `{0, 4, 8, 11}` - 8 = Minor Major 7th `{0, 3, 7, 11}` - 9 = Diminished Major 7th `{0, 3, 6, 11}` - 10 = Major 6th `{0, 4, 7, 9}` - 11 = Minor 6th `{0, 3, 7, 9}` - 12 = 7sus4 `{0, 5, 7, 10}`

N.CS

- **`N.CS r s d c`**

The `N.CS` OP lets you retrieve N table values according to traditional western scales and chords. s, c and d wrap to their ranges automatically and support negative indexing.

Chord Scales - Refer to chord indices in `N.C` OP - `0` = Major `{0, 1, 1, 0, 2, 1, 6}` - 1 = Natural Minor `{1, 6, 0, 1, 1, 0, 2}` - 2 = Harmonic Minor `{8, 6, 7, 1, 2, 0, 3}` - 3 = Melodic Minor `{8, 1, 7, 2, 2, 6, 6}` - 4 = Dorian `{1, 1, 0, 2, 1, 6, 0}` - 5 = Phrygian `{1, 0, 2, 1, 6, 0, 1}` - 6 = Lydian `{0, 2, 1, 6, 0, 1, 1}` - 7 = Mixolydian `{2, 1, 6, 0, 1, 1, 0}` - 8 = Locrian `{6, 0, 1, 1, 0, 2, 1}`

N.B

- **`N.B d/N.B r s`**

Converts a degree in a user-defined equal temperament scale to a value usable by the CV outputs. Default values of `r` and `s` are 0 and R101011010101, corresponding to C-major. To make it easier to generate bit-masks in code, LSB (bit 0) represent the first note in the octave. To avoid having to mirror scales in our heads when entering them by hand, we use `R...` (reverse binary) instead of `B...` (binary).

The bit-masks uses the 12 lower bits.

Note that N.B is using scale at index 0 as used by N.BX ,so N.B and N.BX 0 are equivalent.

Examples:

```
CV 1 N.B 1           ==>  set CV 1 to 1st degree of default scale
                            (C, value corresponding to N 0)
N.B 0 R101011010101   ==>  set scale to C-major (default)
CV 1 N.B 1            ==>  set CV 1 get 1st degree of scale
                            (C, value corresponding to N 0)
N.B 2 R101011010101   ==>  set scale to D-major
CV 1 N.B 3            ==>  set CV 1 to 3rd degree of scale
                            (F#, value corresponding to N 6)
N.B 3 R100101010010   ==>  set scale to Eb-minor pentatonic
CV 1 N.B 2            ==>  set CV 1 to 2nd degree of scale
                            (Gb, value corresponding to N 6)
N.B 5 -3              ==>  set scale to F-lydian using preset
```

Values of s less than 1 sets the bit mask to a preset scale:

```
0:   Ionian (major)
-1:  Dorian
-2:  Phrygian
-3:  Lydian
-4:  Mixolydian
-5:  Aeolean (natural minor)
-6:  Locrian
-7:  Melodic minor
-8:  Harmonic minor
-9:  Major pentatonic
-10: Minor pentatonic
-11  Whole note (1st Messiaen mode)
```

```
-12  Octatonic (half-whole, 2nd Messiaen mode)
-13  Octatonic (whole-half)
-14  3rd Messiaen mode
-15  4th Messiaen mode
-16  5th Messiaen mode
-17  6th Messiaen mode
-18  7th Messiaen mode
-19  Augmented
```

N.BX

- **N.BX i d / N.BX i r s**

Multi-index version of N.B. Index `i` in the range 0-15, allows working with 16 independent scales. Scale at `i` 0 is shared with N.B.

Examples:

```
N.BX 0 0 R101011010101    ==>   set scale at index 0 to C-
major (default)
CV 1 N.BX 0 1            ==>  set CV 1 to 1st degree of scale
                              (C, value corresponding to N 0)
N.BX 1 3 R100101010010    ==>   set scale at index 1 to Eb-
minor pentatonic
CV 1 N.BX 1 2            ==>  set CV 1 to 2nd degree of scale
                              (Gb, value corresponding to N 6)
N.BX 2 5 -3                ==>   set scale at index 2 to F-
lydian using preset
```

QT.CS

- **QT.CS x r s d c**

Quantize 1V/OCT signal x to chord c (1-7) from scale s (0-8, reference N.S scales) at degree d (1-7) with root 1V/OCT pitch `r`.

Chords (1-7) - 1 = Tonic - 2 = Third - 3 = Triad - 4 = Seventh - etc.

Rhythm

Mathematical calculations and tables helpful for rhythmic decisions.

OP	OP *(set)*	*(aliases)*	Description
`BPM x`			milliseconds per beat in BPM `x`
`DR.P b p s`			Drum pattern helper, `b` is the drum bank (`0-4`), `p` is the pattern (0-215) and step is the step number (0-15), returns `0` or 1
`DR.T b p q l s`			Tresillo helper, `b` is the drum bank (`0-4`), `p` is first pattern (0-215), `q` is the second pattern (0-215), `l` is length (`1-64`), and step is the step number (0-length-1), returns `0` or 1
`DR.V p s`			Velocity helper. `p` is the pattern (0-19). `s` is the step number (0-15)
`ER f l i`			Euclidean rhythm, `f` is fill (`1-32`), `l` is length (`1-32`) and `i` is step (any value), returns `0` or 1
`NR p m f s`			Numeric Repeater, `p` is prime pattern (`0-31`), `m` is & mask (`0-3`), `f` is variation factor (`0-16`) and `s` is step (`0-15`), returns `0` or 1

DR.P

- **`DR.P b p s`**

The drum helper uses preset drum patterns to give 16-step gate patterns. Gates wrap after step 16. Bank 0 is a set of pseudo random gates increasing in density at higher numbered patterns, where pattern 0 is empty, and pattern 215 is 1s. Bank 1 is bass drum patterns. Bank 2 is snare drum patterns. Bank 3 is closed hi-hats. Bank 4 is open hi-hits and in some cases cymbals. Bank 1-4 patterns are related to each other (bank 1 pattern 1's bass drum pattern fits bank 2 pattern 1's snare drum pattern). The patterns are from Paul Wenzel's "Pocket Operations" book[22].

DR.T

- **`DR.T b p q l s`**

The Tresillo helper uses the preset drum patterns described in the drum pattern help function in a 3, 3, 2 rythmic formation. In the tresillo, pattern 1 will be repeated twice for

[22] https://shittyrecording.studio/

a number of steps determined by the overall length of the pattern. A pattern of length 8 will play the first three steps of your selected pattern 1 twice, and the first two steps of pattern 2 once. A pattern length of 16 will play the first six steps of selected pattern 1 twice, and the first four steps of pattern 2 once. And so on. The max length is 64. Length will be rounded down to the nearest multiple of 8. The step number wraps at the given length.

DR.V

- **`DR.V p s`**

The velocity helper gives velocity values (0-16383) at each step. The values are intended to be used for drum hit velocities. There are 16 steps, which wrap around. Divide by 129 to convert to midi cc values.

ER

- **`ER f l i`**

Euclidean rhythm helper, as described by Godfried Toussaint in his 2005 paper "The Euclidean Algorithm Generates Traditional Musical Rhythms"[23][24]. From the abstract:

- `f` is fill (1-32) and should be less then or equal to length
- `l` is length (1-32)
- `i` is the step index, and will work with negative as well as positive numbers

If you wish to add rotation as well, use the following form:

`ER f l SUB i r`

where `r` is the number of step of *forward* rotation you want.

For more info, see the post on samdoshi.com[25]

NR

- **`NR p m f s`**

Numeric Repeater is similar to ER, except it generates patterns using the binary arithmetic process found in "Noise Engineering's Numeric Repetitor"[26]. From the description:

Numeric Repetitor is a rhythmic gate generator based on binary arithmetic. A core pattern forms the basis and variation is achieved by treating this pattern as a binary number and multiplying it by another. NR contains 32 prime rhythms derived by examining all possible rhythms and weeding out bad ones via heuristic.

[23] http://cgm.cs.mcgill.ca/~godfried/publications/banff.pdf

[24] Toussaint, G. T. (2005, July). The Euclidean algorithm generates traditional musical rhythms. *In Proceedings of BRIDGES: Mathematical Connections in Art, Music and Science* (pp. 47-56).

[25] http://samdoshi.com/post/2016/03/teletype-euclidean/

[26] https://www.noiseengineering.us/shop/numeric-repetitor

All parameters wrap around their specified ranges automatically and support negative indexing.

Masks - `0` is no mask - 1 is `0x0F0F` - 2 is `0xF003` - 3 is `0x1F0`

For further detail “see the manual”[27].

[27] https://static1.squarespace.com/static/58c709192e69cf2422026fa6/t/5e6041ad4cbc0979d6d793f2/1583366574430/NR_manual.pdf

Metronome

An internal metronome executes the M script at a specified rate (in ms). By default the metronome is enabled (`M.ACT 1`) and set to 1000ms (`M 1000`). The metro can be set as fast as 25ms (`M 25`). An additional `M!` op allows for setting the metronome to experimental rates as high as 2ms (`M! 2`). **WARNING**: when using a large number of i2c commands in the M script at metro speeds beyond the 25ms teletype stability issues can occur.

Access the M script directly with `alt-<F10>` or run the script once using `<F10>`.

OP	OP *(set)*	*(aliases)*	Description
M	**M x**		get/set metronome interval to x (in ms), default 1000, minimum value 25
M!	**M! x**		get/set metronome to experimental interval x (in ms), minimum value 2
M.ACT	**M.ACT x**		get/set metronome activation to x (0/1), default 1 (enabled)
M.RESET			hard reset metronome count without triggering

Randomness

OP	OP *(set)*	*(aliases)*	Description
RAND x		**RND**	generate a random number between 0 and x inclusive
RRAND x y		**RRND**	generate a random number between x and y inclusive
TOSS			randomly return 0 or 1
R	**R x**		get a random number/set R.MIN and R.MAX to same value x (effectively allowing R to be used as a global variable)
R.MIN x			set the lower end of the range from -32768 – 32767, default: 0
R.MAX x			set the upper end of the range from -32768 – 32767, default: 16383
CHAOS x			get next value from chaos generator, or set the current value
CHAOS.R x			get or set the R parameter for the CHAOS generator
CHAOS.ALG x			get or set the algorithm for the CHAOS generator. 0 = LOGISTIC, 1 = CUBIC, 2 = HENON, 3 = CELLULAR
DRUNK	**DRUNK x**		changes by -1, 0, or 1 upon each read saving its state, setting will give it a new value for the next read
DRUNK.MIN	**DRUNK.MIN x**		set the lower bound for DRUNK, default 0
DRUNK.MAX	**DRUNK.MAX x**		set the upper bound for DRUNK, default 255
DRUNK.WRAP	**DRUNK.WRAP x**		should DRUNK wrap around when it reaches it's bounds, default 0
SEED	**SEED x**		get / set the random number generator seed for all SEED ops
RAND.SEED	**RAND.SEED x**	**RAND.SD**, **R.SD**	get / set the random number generator seed for R, RRAND, and RAND ops
TOSS.SEED	**TOSS.SEED x**	**TOSS.SD**	get / set the random number generator seed for the TOSS op

OP	OP *(set)*	*(aliases)*	Description
PROB.SEED	**PROB.SEED x**	**PROB.SD**	get / set the random number generator seed for the PROB mod
DRUNK.SEED	**DRUNK.SEED x**	**DRUNK.SD**	get / set the random number generator seed for the DRUNK op
P.SEED	**P.SEED x**	**P.SD**	get / set the random number generator seed for the P.RND and PN.RND ops

DRUNK

- **DRUNK** / **DRUNK x**

Changes by -1, 0, or 1 upon each read, saving its state. Setting DRUNK will give it a new value for the next read, and drunkedness will continue on from there with subsequent reads.

Setting DRUNK.MIN and DRUNK.MAX controls the lower and upper bounds (inclusive) that DRUNK can reach. DRUNK.WRAP controls whether the value can wrap around when it reaches it's bounds.

Control flow

OP	OP *(set)*	*(aliases)*	Description
IF x: ...			if x is not zero execute command
ELIF x: ...			if all previous IF / ELIF fail, and x is not zero, execute command
ELSE: ...			if all previous IF / ELIF fail, excute command
L x y: ...			run the command sequentially with I values from x to y
W x: ...			run the command while condition x is true
EVERY x: ...		**EV**	run the command every x times the command is called
SKIP x: ...			run the command every time except the xth time.
OTHER: ...			runs the command when the previous EVERY/SKIP did not run its command.
SYNC x			synchronizes *all* EVERY and SKIP counters to offset x.
PROB x: ...			potentially execute command with probability x (0-100)
SCRIPT	**SCRIPT x**	**$**	get current script number, or execute script x (1-10), recursion allowed
SCRIPT.POL x	**SCRIPT.POL x p**	**$.POL**	get script x trigger polarity, or set polarity p (1 rising edge, 2 falling, 3 both)
$F script			execute script as a function
$F1 script param			execute script as a function with 1 parameter
$F2 script param1 param2			execute script as a function with 2 parameters
$L script line			execute script line
$L1 script line param			execute script line as a function with 1 parameter
$L2 script line param1 param2			execute script line as a function with 2 parameters
$S line			execute script line within the same script as a function
$S1 line param			execute script line within the same script as a function with 1 parameter

OP	OP *(set)*	*(aliases)*	Description
$S2 line param1 param2			execute script line within the same script as a function with 2 parameters
I1			get the first parameter when executing a script as a function
I2			get the second parameter when executing a script as a function
FR	**FR x**		get/set the return value when a script is called as a function
SCENE	**SCENE x**		get the current scene number, or load scene x (0-31)
SCENE.G x			load scene x (0-31) without loading grid control states
SCENE.P x			load scene x (0-31) without loading pattern state
KILL			clears stack, clears delays, cancels pulses, cancels slews, disables metronome
BREAK		**BRK**	halts execution of the current script
INIT			clears all state data
INIT.CV x			clears all parameters on CV associated with output x
INIT.CV.ALL			clears all parameters on all CV's
INIT.DATA			clears all data held in all variables
INIT.P x			clears pattern number x
INIT.P.ALL			clears all patterns
INIT.SCENE			loads a blank scene
INIT.SCRIPT x			clear script number x
INIT.SCRIPT.ALL			clear all scripts
INIT.TIME x			clear time on trigger x
INIT.TR x			clear all parameters on trigger x
INIT.TR.ALL			clear all triggers

IF

- **IF x: ...**

If x is not zero execute command

Advanced IF / ELIF / ELSE usage

1. Intermediate statements always run

```text
SCRIPT 1:
IF 0: 0         => do nothing
TR.P 1          => always happens
ELSE: TR.P 2    => else branch runs because of the previous IF
```

2. ELSE without an IF

```text
SCRIPT 1:
ELSE: TR.P 1    => never runs, as there is no preceding IF
```

3. ELIF without an IF

```text
SCRIPT 1:
ELIF 1: TR.P 1  => never runs, as there is no preceding IF
```

4. Independent scripts

```text
SCRIPT 1:
IF 1: TR.P 1    => pulse output 1

SCRIPT 2:
ELSE: TR.P 2   => never runs regardless of what happens in script 1
                  (see example 2)
```

5. Dependent scripts

```text
SCRIPT 1:
IF 0: TR.P 1    => do nothing
SCRIPT 2        => will pulse output 2

SCRIPT 2:
ELSE: TR.P 2    => will not pulse output 2 if called directly,
                   but will if called from script 1
```

L

- **L x y: ...**

Run the command sequentially with I values from x to y.

For example:

```
L 1 4: TR.PULSE I   => pulse outputs 1, 2, 3 and 4
L 4 1: TR.PULSE I   => pulse outputs 4, 3, 2 and 1
```

W

- **W x: ...**

Runs the command while the condition x is true or the loop iterations exceed 10000.

For example, to find the first iterated power of 2 greater than 100:

```
A 2
W LT A 100: A * A A
```

A will be 256.

EVERY

- **EVERY x: ...**
- *alias*: **EV**

Runs the command every x times the line is executed. This is tracked on a per-line basis, so each script can have 6 different “dividers”.

Here is a 1-script clock divider:

```
EVERY 2: TR.P 1
EVERY 4: TR.P 2
EVERY 8: TR.P 3
EVERY 16: TR.P 4
```

The numbers do *not* need to be evenly divisible by each other, so there is no problem with:

```
EVERY 2: TR.P 1
EVERY 3: TR.P 2
```

SKIP

- **SKIP x: ...**

This is the corollary function to EVERY, essentially behaving as its exact opposite.

OTHER

- **OTHER: ...**

OTHER can be used to do somthing alternately with a preceding EVERY or SKIP command.

For example, here is a script that alternates between two triggers to make a four-on-the-floor beat with hats between the beats:

```
EVERY 4: TR.P 1
OTHER: TR.P 2
```

You could add snares on beats 2 and 4 with:

```
SKIP 2: TR.P 3
```

SYNC

- **SYNC x**

Causes all of the EVERY and SYNC counters to synchronize their offsets, respecting their individual divisor values.

Negative numbers will synchronize to to the divisor value, such that `SYNC -1` causes all every counters to be 1 number before their divisor, causing each EVERY to be true on its next call, and each SKIP to be false.

SCRIPT

- **SCRIPT** / **SCRIPT x**
- *alias*: **$**

Execute script x (1-10, 9 = metro, 10 = init), recursion allowed.

There is a limit of 8 for the maximum number of nested calls to SCRIPT to stop infinite loops from locking up the Teletype.

SCRIPT.POL

- **SCRIPT.POL x** / **SCRIPT.POL x p**
- *alias*: **$.POL**

Get or set the trigger polarity of script x, determining which trigger edges the script will fire on.

1: rising edge (default) 2: falling edge 3: either edge

$F

- **$F script**

This op will execute a script similarly to SCRIPT op but it will also return a value, which means you can define a script that calculates something and then use it in an expression. To set the return value, either place an expression at the end of the script without assigning it to anything or assign it to the special function return variable FR. If you do both, FR will be used, and if you don't do either, zero will be returned.

Let's say you update script 1 to return the square of X: `* X X` (which you could also write as `FR * X X`). Then you can use it in an expression like this: `A + A $F 1`.

This op can save space if you have a calculation that is used in multiple places. Other than returning a value, a function script isn't different from a regular script and can perform other actions in addition to calculating something, including calling other scripts. The same limit of 8 maximum nested calls applies here to prevent infinite loops.

If you need to be able to pass parameters into your function, use $F1 or $F2 ops.

$F1

- **`$F1 script param`**

Same as $F but you can also pass a single parameter into the function. Inside the function script you can get the parameter using `I1` op.

Let's say you create a script that returns the square of the passed parameter: `FR * I1 I1`. You can then calculate the square of a number by executing `$F1 value`.

See the description of $F op for more details on executing scripts as functions.

$F2

- **`$F2 script param1 param2`**

Same as $F but you can also pass two parameters into a function. Inside the function script you can get them using `I1` and `I2` ops.

Let's say you create a script that returns a randomly selected value out of the two provided values: `FR ? TOSS I1 I2`. You can then save space by using `$F2 1 X Y` instead of `? TOSS X Y`. More importantly, you could use it in multiple places, and if you later want to change the calculation to something else, you just need to update the function script.

See the description of $F op for more details on executing scripts as functions.

$L

- **`$L script line`**

This op executes the specified script line. This allows you to use a script as a library of sorts, where each line does something different, so you can use the same script for multiple purposes. It also allows you to use free lines in a script to extend another script.

This op behaves similarly to $F op in that it can be used as a function in an expression by setting the return value with FR. Let's say the first line in script 1 is this: `FR * X X`. You can then get the square of X by executing `$L 1 1`.

If you want to use it as a function and you need to pass some parameters into it, use $L1 / $L2 ops.

This op is also useful if you have a loop that doesn't fit on one line - define the line later in the script and then reference it in the loop:

```
#1
L 1 6: A + A $L 1 3
BREAK
SCALE X Y C D I
```

Don't forget to add BREAK before the line so that it's not executed when the whole script is executed. If you use this technique, you can also save space by using $S op which executes a line within the same script.

$L1

- **$L1 script line param**

Execute the specified script line as a function that takes 1 parameter. See the description of $L and $F1 ops for more details.

$L2

- **$L2 script line param1 param2**

Execute the specified script line as a function that takes 2 parameters. See the description of $L and $F2 ops for more details.

$S

- **$S line**

This is exactly the same as `$L $ line` but saves you space on not having to specify the script number if the line you want to execute is within the same script.

See the description of $L for more details.

$S1

- **$S1 line param**

This is exactly the same as `$L1 $ line param` but saves you space on not having to specify the script number if the line you want to execute is within the same script.

See the description of $L1 for more details.

$S2

- **$S2 line param1 param2**

This is exactly the same as `$L2 $ line param1 param2` but saves you space on not having to specify the script number if the line you want to execute is within the same script.

See the description of $L2 for more details.

I1

- **I1**

This op returns the first parameter when a script is called as a function using $F1 / $F2 / $L1 / $L2 / $S1 / $S2 ops. If the script is called using other ops, this op will return zero.

I2

- **I2**

This op returns the second parameter when a script is called as a function using $F2 / $L2 / $S2 ops. If the script is called using other ops, this op will return zero.

FR

- **FR** / **FR x**

Use this op to get or set the return value in a script that is called as a function.

SCENE

- **SCENE** / **SCENE x**

Load scene x (0-31).

Does *not* execute the `I` script. Will *not* execute from the `I` script on scene load. Will execute on subsequent calls to the `I` script.

WARNING: You will lose any unsaved changes to your scene.

SCENE.G

- **SCENE.G x**

Load scene x (0-31) without loading grid button and fader states.

WARNING: You will lose any unsaved changes to your scene.

SCENE.P

- **SCENE.P x**

Load scene x (0-31) without loading pattern data.

WARNING: You will lose any unsaved changes to your scene.

INIT

- **INIT**

WARNING: You will lose all settings when you initialize using INIT - there is NO undo!

INIT.DATA

- **INIT.DATA**

Clears the following variables and resets them to default values: A, B, C, D, CV slew, Drunk min/max, M, O, Q, R, T, TR. Does not affect the CV input (IN) or the Parameter knob (PARAM) values.

Maths

Logical operators such as EQ, 0R and LT return 1 for true, and 0 for false.

OP	OP *(set)*	*(aliases)*	Description
ADD x y		**+**	add x and y together
SUB x y		**-**	subtract y from x
MUL x y		*****	multiply x and y together
DIV x y		**/**	divide x by y
MOD x y		**%**	find the remainder after division of x by y
? x y z			if condition x is true return y, otherwise return z
MIN x y			return the minimum of x and y
MAX x y			return the maximum of x and y
LIM x y z			limit the value x to the range y to z inclusive
WRAP x y z		**WRP**	limit the value x to the range y to z inclusive, but with wrapping
QT x y			round x to the closest multiple of y (quantise)
AVG x y			the average of x and y
EQ x y		**==**	does x equal y
NE x y		**!=** , **XOR**	x is not equal to y
LT x y		**<**	x is less than y
GT x y		**>**	x is greater than y
LTE x y		**<=**	x is less than or equal to y
GTE x y		**>=**	x is greater than or equal to y
INR l x h		**><**	x is greater than l and less than h (within range)
OUTR l x h		**<>**	x is less than l or greater than h (out of range)
INRI l x h		**>=<**	x is greater than or equal to l and less than or equal to h (within range, inclusive)
OUTRI l x h		**<=>**	x is less than or equal to l or greater than or equal to h (out of range, inclusive)
EZ x		**!**	x is 0, equivalent to logical NOT
NZ x			x is not 0
LSH x y		**<<**	left shift x by y bits, in effect multiply x by 2 to the power of y

OP	OP *(set)*	*(aliases)*	Description
RSH x y		**>>**	right shift x by y bits, in effect divide x by 2 to the power of y
LROT x y		**<<<**	circular left shift x by y bits, wrapping around when bits fall off the end
RROT x y		**>>>**	circular right shift x by y bits, wrapping around when bits fall off the end
| x y			bitwise or x
& x y			bitwise and x & y
^ x y			bitwise xor x ^ y
~ x			bitwise not, i.e.: inversion of x
BSET x y			set bit y in value x
BGET x y			get bit y in value x
BCLR x y			clear bit y in value x
BTOG x y			toggle bit y in value x
BREV x			reverse bit order in value x
ABS x			absolute value of x
AND x y		**&&**	logical AND of x and y
AND3 x y z		**&&&**	logical AND of x, y and z
AND4 x y z a		**&&&&**	logical AND of x, y, z and a
OR x y		**||**	logical OR of x and y
OR3 x y z		**|||**	logical OR of x, y and z
OR4 x y z a		**||||**	logical OR of x, y, z and a
SCALE a b x y i		**SCL**	scale `i` from range a to b to range x to y, i.e. `i * (y - x) / (b - a)`
SCALE a b i		**SCL0**	scale `i` from range `0` to a to range `0` to b
EXP x			exponentiation table lookup. `0-16383` range (V `0-10`)
SGN x			sign function: 1 for positive, -1 for negative, 0 for 0

MUL

- **MUL x y**
- *alias*: *****

returns x times y, bounded to integer limits

QT

- **`QT x y`**

Round x to the closest multiple of y. *See also: `QT.S`, `QT.CS`, `QT.B`, `QT.BX` in the Pitch section*.

AND

- **`AND x y`**
- *alias*: **`&&`**

Logical AND of x and y. Returns 1 if both x and y are greater than `0`, otherwise it returns `0`.

AND3

- **`AND3 x y z`**
- *alias*: **`&&&`**

Logical AND of x, y and z. Returns 1 if both x, y and z are greater than `0`, otherwise it returns `0`.

AND4

- **`AND4 x y z a`**
- *alias*: **`&&&&`**

Logical AND of x, y, z and a. Returns 1 if both x, y, z and a are greater than `0`, otherwise it returns `0`.

OR

- **`OR x y`**
- *alias*: `||`

Logical OR of x and y. Returns 1 if either x or y are greater than `0`, otherwise it returns `0`.

OR3

- **`OR3 x y z`**
- *alias*: `|||`

Logical OR of x, y and z. Returns 1 if either x, y or z are greater than `0`, otherwise it returns `0`.

OR4

- **OR4 x y z a**
- *alias*: | | | |

Logical OR of x, y, z and a. Returns 1 if either x, y, z or a are greater than 0, otherwise it returns 0.

Delay

The DEL delay op allow commands to be sheduled for execution after a defined interval by placing them into a buffer which can hold up to 64 commands. Commands can be delayed by up to 16 seconds.

In LIVE mode, the second icon (an upside-down U) will be lit up when there is a command in the DEL buffer.

OP	OP *(set)*	*(aliases)*	Description
`DEL x: ...`			Delay command by x ms
`DEL.CLR`			Clear the delay buffer
`DEL.X x delay_time: ...`			Delay x commands at `delay_time` ms intervals
`DEL.R x delay_time: ...`			Trigger the command following the colon once immediately, and delay `x - 1` commands at `delay_time` ms intervals
`DEL.G x delay_time num denom: ...`			Trigger the command once immediately and `x - 1` times at ms intervals of `delay_time * (num/denom)^n` where n ranges from 0 to `x - 1`.
`DEL.B delay_time bitmask: ...`			Trigger the command up to 16 times at intervals of `delay_time` ms. Active intervals set in 16-bit `bitmask`, LSB = immediate.

DEL

- **`DEL x: ...`**

Delay the command following the colon by x ms by placing it into a buffer. The buffer can hold up to 16 commands. If the buffer is full, additional commands will be discarded.

DEL.CLR

- **`DEL.CLR`**

Clear the delay buffer, cancelling the pending commands.

DEL.X

- **`DEL.X x delay_time: ...`**

Delay the command following the colon x times at intervals of delay_time ms by placing it into a buffer. The buffer can hold up to 16 commands. If the buffer is full, additional commands will be discarded.

DEL.R

- **DEL.R x delay_time: ...**

Delay the command following the colon once immediately, and x - 1 times at intervals of delay_time ms by placing it into a buffer. The buffer can hold up to 16 commands. If the buffer is full, additional commands will be discarded.

DEL.G

- **DEL.G x delay_time num denom: ...**

Trigger the command once immediately and x - 1 times at ms intervals of delay_time * (num/denom)^n where n ranges from 0 to x - 1 by placing it into a buffer. The buffer can hold up to 16 commands. If the buffer is full, additional commands will be discarded.

Stack

These operators manage a last in, first out, stack of commands, allowing them to be memorised for later execution at an unspecified time. The stack can hold up to 8 commands. Commands added to a full stack will be discarded.

OP	OP *(set)*	*(aliases)*	Description
`S: ...`			Place a command onto the stack
`S.CLR`			Clear all entries in the stack
`S.ALL`			Execute all entries in the stack
`S.POP`			Execute the most recent entry
`S.L`			Get the length of the stack

S

- **`S: ...`**

Add the command following the colon to the top of the stack. If the stack is full, the command will be discarded.

S.CLR

- **`S.CLR`**

Clear the stack, cancelling all of the commands.

S.ALL

- **`S.ALL`**

Execute all entries in the stack (last in, first out), clearing the stack in the process.

S.POP

- **`S.POP`**

Pop the most recent command off the stack and execute it.

S.L

- **`S.L`**

Get the number of entries in the stack.

Patterns

Patterns facilitate musical data manipulation– lists of numbers that can be used as sequences, chord sets, rhythms, or whatever you choose. Pattern memory consists four banks of 64 steps. Functions are provided for a variety of pattern creation, transformation, and playback.

New in teletype 2.0, a second version of all Pattern ops have been added. The original P ops (P, P.L, P.NEXT, etc.) act upon the 'working pattern' as defined by P.N. By default the working pattern is assigned to pattern 0 (P.N 0), in order to execute a command on pattern 1 using P ops you would need to first reassign the working pattern to pattern 1 (P.N 1).

The new set of ops, PN (PN, PN.L, PN.NEXT, etc.), include a variable to designate the pattern number they act upon, and don't effect the pattern assignment of the 'working pattern' (ex: PN.NEXT 2 would increment pattern 2 one index and return the value at the new index). For simplicity throughout this introduction we will only refer to the P ops, but keep in mind that they now each have a PN counterpart (all of which are detailed below)

Both patterns and their arrays of numbers are indexed from 0. This makes the first pattern number 0, and the first value of a pattern is index 0. The pattern index (P.I) functions like a playhead which can be moved throughout the pattern and/or read using ops: P, P.I, P.HERE, P.NEXT, and P.PREV. You can contain pattern movements to ranges of a pattern and define wrapping behavior using ops: P.START, P.END, P.L, and P.WRAP.

Values can be edited, added, and retrieved from the command line using ops: P, P.INS, P.RM, P.PUSH, P.HERE, P.NEXT, and P.PREV. Some of these ops will additionally impact the pattern length upon their execution: P.INS, P.RM, P.PUSH, and P.POP.

To see your current pattern data use the <tab> key to cycle through live mode, edit mode, and pattern mode. In pattern mode each of the 4 patterns is represented as a column. You can use the arrow keys to navigate throughout the 4 patterns and their 64 values. For reference a key of numbers runs the down the lefthand side of the screen in pattern mode displaying 0-63.

From a blank set of patterns you can enter data by typing into the first cell in a column. Once you hit <enter> you will move to the cell below and the pattern length will become one step long. You can continue this process to write out a pattern of desired length. The step you are editing is always the brightest. As you add steps to a pattern by editing the value and hitting <enter> they become brighter than the unused cells. This provides a visual indication of the pattern length.

The start and end points of a pattern are represented by the dotted line next to the column, and the highlighted dot in this line indicates the current pattern index for each of the patterns. See the key bindings for an extensive list of editing shortcuts available within pattern mode.

OP	OP *(set)*	*(aliases)*	Description
P.N	**P.N x**		get/set the pattern number for the working pattern, default 0
P x	**P x y**		get/set the value of the working pattern at index x
PN x y	**PN x y z**		get/set the value of pattern x at index y
P.L	**P.L x**		get/set pattern length of the working pattern, non-destructive to data
PN.L x	**PN.L x y**		get/set pattern length of pattern x. non-destructive to data
P.WRAP	**P.WRAP x**		when the working pattern reaches its bounds does it wrap (0/1), default 1 (enabled)
PN.WRAP x	**PN.WRAP x y**		when pattern x reaches its bounds does it wrap (0/1), default 1 (enabled)
P.START	**P.START x**		get/set the start location of the working pattern, default 0
PN.START x	**PN.START x y**		get/set the start location of pattern x, default 0
P.END	**P.END x**		get/set the end location of the working pattern, default 63
PN.END x	**PN.END x y**		get/set the end location of the pattern x, default 63
P.I	**P.I x**		get/set index position for the working pattern.
PN.I x	**PN.I x y**		get/set index position for pattern x
P.HERE	**P.HERE x**		get/set value at current index of working pattern
PN.HERE x	**PN.HERE x y**		get/set value at current index of pattern x
P.NEXT	**P.NEXT x**		increment index of working pattern then get/set value
PN.NEXT x	**PN.NEXT x y**		increment index of pattern x then get/set value
P.PREV	**P.PREV x**		decrement index of working pattern then get/set value
PN.PREV x	**PN.PREV x y**		decrement index of pattern x then get/set value
P.INS x y			insert value y at index x of working pattern, shift later values down, destructive to loop length

OP	OP *(set)*	*(aliases)*	Description
PN.INS x y z			insert value z at index y of pattern x, shift later values down, destructive to loop length
P.RM x			delete index x of working pattern, shift later values up, destructive to loop length
PN.RM x y			delete index y of pattern x, shift later values up, destructive to loop length
P.PUSH x			insert value x to the end of the working pattern (like a stack), destructive to loop length
PN.PUSH x y			insert value y to the end of pattern x (like a stack), destructive to loop length
P.POP			return and remove the value from the end of the working pattern (like a stack), destructive to loop length
PN.POP x			return and remove the value from the end of pattern x (like a stack), destructive to loop length
P.MIN			find the first minimum value in the pattern between the START and END for the working pattern and return its index
PN.MIN x			find the first minimum value in the pattern between the START and END for pattern x and return its index
P.MAX			find the first maximum value in the pattern between the START and END for the working pattern and return its index
PN.MAX x			find the first maximum value in the pattern between the START and END for pattern x and return its index
P.SHUF			shuffle the values in active pattern (between its START and END)
PN.SHUF x			shuffle the values in pattern x (between its START and END)

OP	OP *(set)*	*(aliases)*	Description
P.ROT n			rotate the values in the active pattern n steps (between its START and END, negative rotates backward)
PN.ROT x n			rotate the values in pattern x (between its START and END, negative rotates backward)
P.REV			reverse the values in the active pattern (between its START and END)
PN.REV x			reverse the values in pattern x
P.RND			return a value randomly selected between the start and the end position
PN.RND x			return a value randomly selected between the start and the end position of pattern x
P.+ x y			increase the value of the working pattern at index x by y
PN.+ x y z			increase the value of pattern x at index y by z
P.- x y			decrease the value of the working pattern at index x by y
PN.- x y z			decrease the value of pattern x at index y by z
P.+W x y a b			increase the value of the working pattern at index x by y and wrap it to a..b range
PN.+W x y z a b			increase the value of pattern x at index y by z and wrap it to a..b range
P.-W x y a b			decrease the value of the working pattern at index x by y and wrap it to a..b range
PN.-W x y z a b			decrease the value of pattern x at index y by z and wrap it to a..b range
P.MAP: ...			apply the 'function' to each value in the active pattern, I takes each pattern value
PN.MAP x: ...			apply the 'function' to each value in pattern x, I takes each pattern value

P.N

- **P.N / P.N x**

get/set the pattern number for the working pattern, default 0. All P ops refer to this pattern.

P

- **P x / P x y**

get/set the value of the working pattern at index x. All positive values (0-63) can be set or returned while index values greater than 63 clip to 63. Negative x values are indexed backwards from the end of the pattern length of the working pattern.

Example:

with a pattern length of 6 for the working pattern:

P 10 retrieves the working pattern value at index 6

P.I -2 retrieves the working pattern value at index 4

This applies to PN as well, except the pattern number is the first variable and a second variable specifies the index.

P.WRAP

- **P.WRAP / P.WRAP x**

when the working pattern reaches its bounds does it wrap (0/1). With PN.WRAP enabled (1), when an index reaches its upper or lower bound using P.NEXT or P.PREV it will wrap to the other end of the pattern and you can continue advancing. The bounds of P.WRAP are defined through P.L, P.START, and P.END.

If wrap is enabled (P.WRAP 1) a pattern will begin at its start location and advance to the lesser index of either its end location or the end of its pattern length

Examples:

With wrap enabled, a pattern length of 6, a start location of 2 , and an end location of 8.

```
P.WRAP 1; P.L 6; P.START 2; P.END 8
```

The pattern will wrap between the indexes 2 and 5.

With wrap enabled, a pattern length of 10, a start location of 3, and an end location of 6.

```
P.WRAP 1; P.L 10; P.START 3; P.END 6
```

The pattern will wrap between the indexes 3 and 6.

If wrap is disabled (P.WRAP 0) a pattern will run between its start and end locations and halt at either bound.

This applies to PN.WRAP as well, except the pattern number is the first variable and a second variable specifies the wrap behavior (0/1).

P.I

- **P.I / P.I x**

get/set index position for the working pattern. all values greater than pattern length return the first step beyond the pattern length. negative values are indexed backwards from the end of the pattern length.

Example:

With a pattern length of 6 (P.L 6), yielding an index range of 0-5:

```
P.I 3
```

moves the index of the working pattern to 3

```
P.I 10
```

moves the index of the working pattern to 6

```
P.I -2
```

moves the index of the working pattern to 4

This applies to PN.I, except the pattern number is the first variable and a second variable specifics the index.

P.MAP

- **P.MAP: ...**

Replace each cell in the active pattern (between the START and END of the pattern) by assigning the variable I to the current value of the cell, evaluating the command after the mod, and assigning that pattern cell with the result. The 'map' higher-order function from functional programming, with the command giving the function of I to map over the pattern.

For example:

```
P.MAP: * 2 I  => double each cell in the active pattern
```

Queue

These operators manage a first in, first out, queue of values. The length of the queue can be dynamically changed up to a maximum size of 64 elements. A fixed length can be set with the `Q.N` operator, or the queue can grow and shrink automatically by setting `Q.GRW 1`. The queue contents will be preserved when the length is shortened.

Queues also offer operators that do math on the entire queue (the `Q.AVG` operator is particularly useful for smoothing input values) or copy the queue to and from a tracker pattern.

Most operators manipulates the elements up to (and including) length. Exceptions are `Q.I i x` and `Q.P2`.

Examples, only first 8 elements shown for clarity: By default all elements of the queue have a value of `0` and the length is set to 1.

```
Q.N "length" ->|
element nb: 1  |  2 3 4 5 6 7 8
value       0  |  0 0 0 0 0 0 0
```

Using the Q OP will add values to the beginning of the queue and push the other elements to the right. `Q 1`

```
1  |  0 0 0 0 0 0 0

Q 2 // add 2 to queue
Q 3 // add 3 to queue

3  |  2 1 0 0 0 0 0
```

Using the Q getter OP will return the last element in the queue, but not modify content or the state of the queue.

```
Q // will return 3

3  |  2 1 0 0 0 0 0
```

Using the `Q.N` OP will either return the position of the end marker (1-indexed) or move it:

```
Q.N 2 // increace the length to two by moving the end marker:

3 2  |  1 0 0 0 0 0

Q // get the value at the end, now `2`
```

By default grow is disabled, but it can be turned on with `Q.GRW 1`. With grow enabled, the queue will automatically expand when new elements are added with `Q x` and likewise shrink when reading with Q.

```
Q.GRW // enable grow
3 2  |  1 0 0 0 0 0
Q 4 // add to to queue
```

```
4 3 2  |  1 0 0 0 0
Q // read element from queue, will return 2
4 3  |  2 1 0 0 0 0
```

OP	OP *(set)*	*(aliases)*	Description
Q	**Q x**		Modify the queue entries
Q.N	**Q.N x**		The queue length
Q.AVG	**Q.AVG x**		Return the average of the queue
Q.CLR	**Q.CLR x**		Clear queue
Q.GRW	**Q.GRW x**		Get/set grow state
Q.SUM	**Q.SUM x**		Get sum of elements
Q.MIN	**Q.MIN x**		Get/set minimum value
Q.MAX	**Q.MAX x**		Get/set maximum value
Q.RND	**Q.RND x**		Get random element/randomize elements
Q.SRT	**Q.SRT**		Sort all or part of queue
Q.REV			Reverse queue
Q.SH	**Q.SH x**		Shift elements in queue
Q.ADD x	**Q.ADD x i**		Perform addition on elements in queue
Q.SUB x	**Q.SUB x i**		Perform subtraction on elements in queue
Q.MUL x	**Q.MUL x i**		Perform multiplication on elements in queue
Q.DIV x	**Q.DIV x i**		Perform division on elements in queue
Q.MOD x	**Q.MOD x i**		Perform module (%) on elements in queue
Q.I i	**Q.I i x**		Get/set value of elements at index
Q.2P	**Q.2P i**		Copy queue to current pattern/copy queue to pattern at index `i`
Q.P2	**Q.P2 i**		Copy current pattern to queue/copy pattern at index `i` to queue

Q

- **Q** / **Q x**

Gets the output value from the queue, or places x into the queue.

Q.N

- **Q.N** / **Q.N x**

Gets/sets the length of the queue. The length is 1-indexed.

Q.AVG

- **Q.AVG** / **Q.AVG x**

Getting the value the average of the values in the queue. Setting x sets the value of each entry in the queue to x.

Q.CLR

- **Q.CLR** / **Q.CLR x**

Clear queue, set all values to 0, length to 1. If parameter x is provided, set first elements to x.

Q.GRW

- **Q.GRW** / **Q.GRW x**

If grow is set (value of 1) the queue will automatically grow and shrink when using Q (popping and pushing).

Q.SUM

- **Q.SUM** / **Q.SUM x**

Get sum of all elements in queue.

Q.MIN

- **Q.MIN** / **Q.MIN x**

Get the minimum value of elements in queue. If x is provided, set elements with a value less than x to x.

Q.MAX

- **Q.MAX** / **Q.MAX x**

Get the maximum value of elements in queue. If x is provided, set elements with a value greater than x to x.

Q.RND

- **Q.RND** / **Q.RND x**

Get a random element in queue.

If x > 0, set all elements to a random value 0-x. If x < 0, swap two elements −x number of times. IF x == 0, do nothing.

Q.SRT

- **Q.SRT** / **Q.SRT**

Sort elements in queue. With no arguments, entire queue is sorted in accending order.

If x > 0, sort elements from index `i` to the end of queue. If x < 0, sort elements from beginning of queue to index `-i`. IF x == 0, sort all elements.

Index `i` is 0-indexed.

Q.REV

- **Q.REV**

Reverse order of elements in queue.

Q.SH

- **Q.SH** / **Q.SH x**

Shift elements x locations to right. Negative values of x shifts to the left. No value provided is equal to x = 1. Shifting is wrapped.

Q.ADD

- **Q.ADD x** / **Q.ADD x i**

Add x to all elements in queue. If index `i` is provided, only perform addition on element at index `i`.

Index `i` is 0-indexed.

Q.SUB

- **Q.SUB x** / **Q.SUB x i**

Subtract x from all elements in queue. If index `i` is provided, only perform subtraction on element at index `i`.

Index `i` is 0-indexed.

Q.MUL

- **Q.MUL x** / **Q.MUL x i**

Multiply all elements in queue with x. If index `i` is provided, only perform multiplication on element at index `i`.

Index `i` is 0-indexed.

Q.DIV

- **Q.DIV x** / **Q.DIV x i**

Divide all elements in queue by `x`. If index `i` is provided, only perform division on element at index `i`.

Index `i` is 0-indexed.

Q.MOD

- **Q.MOD x** / **Q.MOD x i**

Perform modulo of x (value = value % x) on all elements in queue. If index `i` is provided, only perform modulo operation on element at index `i`.

Index `i` is 0-indexed.

Q.I

- **Q.I i** / **Q.I i x**

Get value of element at index `i` or set value of element `i` to value `x`. Indexing works on entire lenght of queue, and is not limited to elements below queue end point.

Index `i` is 0-indexed.

Q.2P

- **Q.2P** / **Q.2P i**

Copy entire queue to current pattern or (if `i` provided) pattern at index `i`.

Index `i` is 0-indexed.

Q.P2

- **Q.P2** / **Q.P2 i**

Copy current pattern to queue or (if `i` provided) copy pattern at index `i` to queue.

Index `i` is 0-indexed.

Turtle

A 2-dimensional, movable index into the pattern values as displayed on the TRACKER screen.

OP	OP *(set)*	*(aliases)*	Description
@	**@ x**		get or set the current pattern value under the turtle
@X	**@X x**		get the turtle X coordinate, or set it to x
@Y	**@Y x**		get the turtle Y coordinate, or set it to x
@MOVE x y			move the turtle x cells in the X axis and y cells in the Y axis
@F x1 y1 x2 y2			set the turtle's fence to corners x1,y1 and x2,y2
@FX1	**@FX1 x**		get the left fence line or set it to x
@FX2	**@FX2 x**		get the right fence line or set it to x
@FY1	**@FY1 x**		get the top fence line or set it to x
@FY2	**@FY2 x**		get the bottom fence line or set it to x
@SPEED	**@SPEED x**		get the speed of the turtle's @STEP in cells per step or set it to x
@DIR	**@DIR x**		get the direction of the turtle's @STEP in degrees or set it to x
@STEP			move @SPEED/100 cells forward in @DIR, triggering @SCRIPT on cell change
@BUMP	**@BUMP 1**		get whether the turtle fence mode is BUMP, or set it to BUMP with 1
@WRAP	**@WRAP 1**		get whether the turtle fence mode is WRAP, or set it to WRAP with 1
@BOUNCE	**@BOUNCE 1**		get whether the turtle fence mode is BOUNCE, or set it to BOUNCE with 1
@SCRIPT	**@SCRIPT x**		get which script runs when the turtle changes cells, or set it to x
@SHOW	**@SHOW 0/1**		get whether the turtle is displayed on the TRACKER screen, or turn it on or off

Grid

Grid operators allow creating scenes that can interact with grid connected to teletype (important: grid must be powered externally, do not connect it directly to teletype!). You can light up individual LEDs, draw shapes and create controls (such as buttons and faders) that can be used to trigger and control scripts. You can take advantage of grid operators even without an actual grid by using the built in Grid Visualizer.

For more information on grid integration see Advanced section and Grid Studies[28].

As there are many operators let's review some naming conventions that apply to the majority of them. All grid ops start with `G.`. For control related ops this is followed by 3 letters specifying the control: `G.BTN` for buttons, `G.FDR` for faders. To define a control you use the main ops `G.BTN` and `G.FDR`. To define multiple controls replace the last letter with X: `G.BTX`, `G.FDX`.

All ops that initialize controls use the same list of parameters: id, coordinates, width, height, type, level, script. When creating multiple controls there are two extra parameters: the number of columns and the number of rows. Controls are created in the current group (set with `G.GRP`). To specify a different group use the group versions of the 4 above ops - `G.GBT`, `G.GFD`, `G.GBX`, `G.GFX` and specify the desired group as the first parameter.

All controls share some common properties, referenced by adding a `.` and:

- EN: `G.BTN.EN`, `G.FDR.EN` - enables or disables a control
- V: `G.BTN.V`, `G.FDR.V` - value, 1/0 for buttons, range value for faders
- L: `G.BTN.L`, `G.FDR.L` - level (brightness level for buttons and coarse faders, max value level for fine faders)
- X: `G.BTN.X`, `G.FDR.X` - the X coordinate
- Y: `G.BTN.Y`, `G.FDR.Y` - the Y coordinate

To get/set properties for individual controls you normally specify the control id as the first parameter: `G.FDR.V 5` will return the value of fader 5. Quite often the actual id is not important, you just want to work with the latest control pressed. As these are likely the ops to be used most often they are offered as shortcuts without a `.`: `G.BTNV` returns the value of the last button pressed, `G.FDRL 4` will set the level of the last fader pressed etc etc.

OP	OP *(set)*	*(aliases)*	Description
`G.RST`			full grid reset
`G.CLR`			clear all LEDs
`G.DIM level`			set dim level
`G.ROTATE x`			set grid rotation
`G.KEY x y action`			emulate grid press
`G.GRP`	**`G.GRP id`**		get/set current group

[28] https://github.com/scanner-darkly/teletype/wiki/GRID-INTEGRATION

OP	OP *(set)*	*(aliases)*	Description
G.GRP.EN id	**G.GRP.EN id x**		enable/disable group or check if enabled
G.GRP.RST id			reset all group controls
G.GRP.SW id			switch groups
G.GRP.SC id	**G.GRP.SC id script**		get/set group script
G.GRPI			get last group
G.LED x y	**G.LED x y level**		get/set LED
G.LED.C x y			clear LED
G.REC x y w h fill border			draw rectangle
G.RCT x1 y1 x2 y2 fill border			draw rectangle
G.BTN id x y w h type level script			initialize button
G.GBT group id x y w h type level script			initialize button in group
G.BTX id x y w h type level script columns rows			initialize multiple buttons
G.GBX group id x y w h type level script columns rows			initialize multiple buttons in group
G.BTN.EN id	**G.BTN.EN id x**		enable/disable button or check if enabled
G.BTN.X id	**G.BTN.X id x**		get/set button x coordinate
G.BTN.Y id	**G.BTN.Y id y**		get/set button y coordinate
G.BTN.V id	**G.BTN.V id value**		get/set button value
G.BTN.L id	**G.BTN.L id level**		get/set button level
G.BTNI			id of last pressed button
G.BTNX	**G.BTNX x**		get/set x of last pressed button
G.BTNY	**G.BTNY y**		get/set y of last pressed button
G.BTNV	**G.BTNV value**		get/set value of last pressed button
G.BTNL	**G.BTNL level**		get/set level of last pressed button

OP	OP *(set)*	*(aliases)*	Description
G.BTN.SW id			switch button
G.BTN.PR id action			emulate button press/release
G.GBTN.V group value			set value for group buttons
G.GBTN.L group odd_level even_level			set level for group buttons
G.GBTN.C group			get count of currently pressed
G.GBTN.I group index			get id of pressed button
G.GBTN.W group			get button block width
G.GBTN.H group			get button block height
G.GBTN.X1 group			get leftmost pressed x
G.GBTN.X2 group			get rightmost pressed x
G.GBTN.Y1 group			get highest pressed y
G.GBTN.Y2 group			get lowest pressed y
G.FDR id x y w h type level script			initialize fader
G.GFD grp id x y w h type level script			initialize fader in group
G.FDX id x y w h type level script columns rows			initialize multiple faders
G.GFX group id x y w h type level script columns rows			initialize multiple faders in group
G.FDR.EN id	**G.FDR.EN id x**		enable/disable fader or check if enabled
G.FDR.X id	**G.FDR.X id x**		get/set fader x coordinate
G.FDR.Y id	**G.FDR.Y id y**		get/set fader y coordinate
G.FDR.N id	**G.FDR.N id value**		get/set fader value
G.FDR.V id	**G.FDR.V id value**		get/set scaled fader value
G.FDR.L id	**G.FDR.L id level**		get/set fader level
G.FDRI			id of last pressed fader
G.FDRX	**G.FDRX x**		get/set x of last pressed fader

OP	OP *(set)*	*(aliases)*	Description
G.FDRY	**G.FDRY y**		get/set y of last pressed fader
G.FDRN	**G.FDRN value**		get/set value of last pressed fader
G.FDRV	**G.FDRV value**		get/set scaled value of last pressed fader
G.FDRL	**G.FDRL level**		get/set level of last pressed fader
G.FDR.PR id value			emulate fader press
G.GFDR.N group value			set value for group faders
G.GFDR.V group value			set scaled value for group faders
G.GFDR.L group odd_level even_level			set level for group faders
G.GFDR.RN group min max			set range for group faders

G.RST

- **G.RST**

Full grid reset - hide all controls and reset their properties to the default values, clear all LEDs, reset the dim level and the grid rotation.

G.CLR

- **G.CLR**

Clear all LEDs set with G.LED, G.REC or G.RCT.

G.DIM

- **G.DIM level**

Set the dim level (0..14, higher values dim more). To remove set to 0.

G.ROTATE

- **G.ROTATE x**

Set the grid rotation (0 - no rotation, 1 - rotate by 180 degrees).

G.KEY

- **G.KEY x y action**

Emulate a grid key press at the specified coordinates (0-based). Set `action` to 1 to emulate a press, 0 to emulate a release. You can also emulate a button press with `G.BTN.PR` and a fader press with `G.FDR.PR`.

G.GRP

- **G.GRP** / **G.GRP id**

Get or set the current group. Grid controls created without specifying a group will be assigned to the current group. This op doesn't enable/disable groups - use `G.GRP.EN` for that. The default current group is 0. 64 groups are available.

G.GRP.EN

- **G.GRP.EN id** / **G.GRP.EN id x**

Enable or disable the specified group or check if it's currently enabled. 1 means enabled, 0 means disabled. Enabling or disabling a group enables / disables all controls assigned to that group (disabled controls are not shown and receive no input). This allows groups to be used as pages - initialize controls in different groups, and then simply enable one group at a time.

G.GRP.RST

- **G.GRP.RST id**

Reset all controls associated with the specified group. This will disable the controls and reset their properties to the default values. This will also reset the fader scale range to 0..16383.

G.GRP.SW

- **G.GRP.SW id**

Switch groups. Enables the specified group, disables all others.

G.GRP.SC

- **G.GRP.SC id** / **G.GRP.SC id script**

Assign a script to the specified group, or get the currently assigned script. The script gets executed whenever a control associated with the group receives input. It is possible to have different scripts assigned to a control and the group it belongs to. Use 9 for Metro and 10 for Init. To unassign, set it to 0.

G.GRPI

- **G.GRPI**

Get the id of the last group that received input. This is useful when sharing a script between multiple groups.

G.LED

- **G.LED x y** / **G.LED x y level**

Set the LED level or get the current level at the specified coordinates. Possible level range is 0..15 (on non varibright grids anything below 8 is 'off', 8 or above is 'on').

Grid controls get rendered first, and LEDs are rendered last. This means you can use LEDs to accentuate certain areas of the UI. This also means that any LEDs that are set will block whatever is underneath them, even with the level of 0. In order to completely clear an LED set its level to -3. There are two other special values for brightness: -1 will dim, and -2 will brighten what's underneath. They can be useful to highlight the current sequence step, for instance.

G.LED.C

- **G.LED.C x y**

Clear the LED at the specified coordinates. This is the same as setting the brightness level to -3. To clear all LEDs use `G.CLR`.

G.REC

- **G.REC x y w h fill border**

Draw a rectangle with the specified width and height. `x` and `y` are the coordinates of the top left corner. Coordinates are 0-based, with the 0,0 point located at the top left corner of the grid. You can draw rectangles that are partially outside of the visible area, and they will be properly cropped.

`fill` and `border` specify the brightness levels for the inner area and the one-LED-wide border respectively, 0..15 range. You can use the three special brightness levels: -1 to dim, -2 to brighten and -3 for transparency (you could draw just a frame by setting `fill` to -3, for instance).

To draw lines, set the width or the height to 1. In this case only `border` brightness level is used.

G.RCT

- **G.RCT x1 y1 x2 y2 fill border**

Same as `G.REC` but instead of specifying the width and height you specify the coordinates of the top left corner and the bottom right corner.

G.BTN

- **G.BTN id x y w h type level script**

Initializes and enables a button with the specified id. 256 buttons are available (ids are 0-based so the possible id range is 0..255. The button will be assigned to the current group (set with G.GRP). Buttons can be reinitialized at any point.

x and y specify the coordinates of the top left corner, and w and h specify width and height respectively. type determines whether the button is latching (1) or momentary (0). level sets the "off" brightness level, possible rand is -3..15 (the brightness level for pressed buttons is fixed at 13).

script specifies the script to be executed when the button is pressed or released (the latter only for momentary buttons). Use 9 for Metro and 10 for Init. Use 0 if you don't need a script assigned.

G.GBT

- **G.GBT group id x y w h type level script**

Initialize and enable a button. Same as G.BTN but you can also choose which group to assign the button too.

G.BTX

- **G.BTX id x y w h type level script columns rows**

Initialize and enable a block of buttons in the current group with the specified number of columns and rows . Ids are incremented sequentially by columns and then by rows.

G.GBX

- **G.GBX group id x y w h type level script columns rows**

Initialize and enable a block of buttons. Same as G.BTX but you can also choose which group to assign the buttons too.

G.BTN.EN

- **G.BTN.EN id** / **G.BTN.EN id x**

Enable (set x to 1) or disable (set x to 0) a button with the specified id, or check if it's currently enabled. Disabling a button hides it and stops it from receiving input but keeps all the other properties (size/location etc) intact.

G.BTN.X

- **G.BTN.X id** / **G.BTN.X id x**

Get or set x coordinate for the specified button's top left corner.

G.BTN.Y

- **G.BTN.Y id** / **G.BTN.Y id y**

Get or set y coordinate for the specified button's top left corner.

G.BTN.V

- **G.BTN.V id** / **G.BTN.V id value**

Get or set the specified button's value. For buttons the value of 1 means the button is pressed and 0 means it's not. If there is a script assigned to the button it will not be triggered if you change the value - use `G.BTN.PR` for that.

Button values don't change when a button is disabled. Button values are stored with the scene (both to flash and to USB sticks).

G.BTN.L

- **G.BTN.L id** / **G.BTN.L id level**

Get or set the specified button's brightness level (-3..15). Please note you can only set the level for unpressed buttons, the level for pressed buttons is fixed at 13.

G.BTNI

- **G.BTNI**

Get the id of the last pressed button. This is useful when multiple buttons are assigned to the same script.

G.BTNX

- **G.BTNX** / **G.BTNX x**

Get or set x coordinate of the last pressed button's top left corner. This is the same as `G.BTN.X G.BTNI`.

G.BTNY

- **G.BTNY** / **G.BTNY y**

Get or set y coordinate of the last pressed button's top left corner. This is the same as `G.BTN.Y G.BTNI`.

G.BTNV

- **G.BTNV** / **G.BTNV value**

Get or set the value of the last pressed button. This is the same as `G.BTN.V G.BTNI`. This op is especially useful with momentary buttons when you want to react to presses or releases only - just put `IF EZ G.BTNV: BREAK` in the beginning of the assigned script (this will ignore releases, to ignore presses replace `NZ` with `EZ`).

G.BTNL

- **`G.BTNL` / `G.BTNL level`**

Get or set the brightness level of the last pressed button. This is the same as `G.BTN.L G.BTNI`.

G.BTN.SW

- **`G.BTN.SW id`**

Set the value of the specified button to 1 (pressed), set it to 0 (not pressed) for all other buttons within the same group (useful for creating radio buttons).

G.BTN.PR

- **`G.BTN.PR id action`**

Emulate pressing/releasing the specified button. Set `action` to `1` for press, `0` for release (`action` is ignored for latching buttons).

G.GBTN.V

- **`G.GBTN.V group value`**

Set the value for all buttons in the specified group.

G.GBTN.L

- **`G.GBTN.L group odd_level even_level`**

Set the brightness level (0..15) for all buttons in the specified group. You can use different values for odd and even buttons (based on their index within the group, not their id) - this can be a good way to provide some visual guidance.

G.GBTN.C

- **`G.GBTN.C group`**

Get the total count of all the buttons in the specified group that are currently pressed.

G.GBTN.I

- **G.GBTN.I group index**

Get the id of a currently pressed button within the specified group by its index (0-based). The index should be between 0 and C-1 where C is the total count of all pressed buttons (you can get it using `G.GBTN.C`).

G.GBTN.W

- **G.GBTN.W group**

Get the width of the rectangle formed by pressed buttons within the specified group. This is basically the distance between the leftmost and the rightmost pressed buttons, inclusive. This op is useful for things like setting a loop's length, for instance. To do so, check if there is more than one button pressed (using `G.GBTN.C`) and if there is, use `G.GBTN.W` to set the length.

G.GBTN.H

- **G.GBTN.H group**

Get the height of the rectangle formed by pressed buttons within the specified group (see `G.GBTN.W` for more details).

G.GBTN.X1

- **G.GBTN.X1 group**

Get the X coordinate of the leftmost pressed button in the specified group. If no buttons are currently pressed it will return -1.

G.GBTN.X2

- **G.GBTN.X2 group**

Get the X coordinate of the rightmost pressed button in the specified group. If no buttons are currently pressed it will return -1.

G.GBTN.Y1

- **G.GBTN.Y1 group**

Get the Y coordinate of the highest pressed button in the specified group. If no buttons are currently pressed it will return -1.

G.GBTN.Y2

- **G.GBTN.Y2 group**

Get the Y coordinate of the lowest pressed button in the specified group. If no buttons are currently pressed it will return -1.

G.FDR

- **G.FDR id x y w h type level script**

Initializes and enables a fader with the specified id. 64 faders are available (ids are 0-based so the possible id range is 0..63). The fader will be assigned to the current group (set with G.GRP). Faders can be reinitialized at any point.

x and y specify the coordinates of the top left corner, and w and h specify width and height respectively.

type determines the fader type and orientation. Possible values are:

- 0 - coarse, horizontal bar
- 1 - coarse, vertical bar
- 2 - coarse, horizontal dot
- 3 - coarse, vertical dot
- 4 - fine, horizontal bar
- 5 - fine, vertical bar
- 6 - fine, horizontal dot
- 7 - fine, vertical dot

Coarse faders have the possible range of 0..N-1 where N is width for horizontal faders or height for vertical faders. Pressing anywhere within the fader area sets the fader value accordingly. Fine faders allow selecting a bigger range of values by mapping the range to the fader's height or width and dedicating the edge buttons for incrementing/decrementing. Fine faders employ varibrightness to reflect the current value.

level has a different meaning for coarse and fine faders. For coarse faders it selects the background brightness level (similar to buttons). For fine faders this is the maximum value level (the minimum level being 0). In order to show each value distinctly using varibright the maximum level possible is the number of available buttons multiplied by 16 minus 1 (since range is 0-based). Remember that 2 buttons are always reserved for increment/decrement. Using a larger number is allowed - it will be automatically adjusted to what's possible.

script specifies the script to be executed when the fader value is changed. Use 9 for Metro and 10 for Init. Use 0 if you don't need a script assigned.

G.GFD

- **G.GFD grp id x y w h type level script**

Initialize and enable a fader. Same as G.FDR but you can also choose which group to assign the fader too.

G.FDX

- **G.FDX id x y w h type level script columns rows**

Initialize and enable a block of faders with the specified number of columns and rows in the current group. Ids are incremented sequentially by columns and then by rows.

G.GFX

- **G.GFX group id x y w h type level script columns rows**

Initialize and enable a block of faders. Same as G.FDX but you can also choose which group to assign the faders too.

G.FDR.EN

- **G.FDR.EN id** / **G.FDR.EN id x**

Enable (set x to 1) or disable (set x to 0) a fader with the specified id, or check if it's currently enabled. Disabling a fader hides it and stops it from receiving input but keeps all the other properties (size/location etc) intact.

G.FDR.X

- **G.FDR.X id** / **G.FDR.X id x**

Get or set x coordinate for the specified fader's top left corner.

G.FDR.Y

- **G.FDR.Y id** / **G.FDR.Y id y**

Get or set y coordinate for the specified fader's top left corner.

G.FDR.N

- **G.FDR.N id** / **G.FDR.N id value**

Get or set the specified fader's value. The possible range for coarse faders is 0..N-1 where N is fader's width (for horizontal faders) or height (for vertical faders). For fine faders the possible range is 0..N where N is the maximum level set when the fader was initialized (see G.FDR for more details).

Sometimes it's more convenient to map the possible fader range to a different range (when using it to control a CV, for instance). Use G.FDR.V for that.

If there is a script assigned to the fader it will not be triggered if you change the value - use G.FDR.PR for that.

Fader values don't change when a fader is disabled. Fader values are stored with the scene (both to flash and to USB sticks).

G.FDR.V

- **G.FDR.V id** / **G.FDR.V id value**

Get or set the specified fader's value mapped to a range set with G.GFDR.RN. This op is very convenient for using faders to control a known range, such as CV - simply create a fader and set a range and then assign values directly without any additional calculations, like this: `CV 1 G.FDR.V 1`.

G.FDR.L

- **G.FDR.L id** / **G.FDR.L id level**

Get or set the specified fader's brightness level (for coarse faders), or the maximum value level (for fine faders).

G.FDRI

- **G.FDRI**

Get the id of the last pressed fader. This is useful when multiple faders are assigned to the same script.

G.FDRX

- **G.FDRX** / **G.FDRX x**

Get or set x coordinate of the last pressed fader's top left corner. This is the same as `G.FDR.X G.FDRI`.

G.FDRY

- **G.FDRY** / **G.FDRY y**

Get or set y coordinate of the last pressed fader's top left corner. This is the same as `G.BTN.Y G.BTNI`.

G.FDRN

- **G.FDRN** / **G.FDRN value**

Get or set the value of the last pressed fader. This is the same as `G.FDR.N G.FDRI`. See G.FDR.N for more details.

G.FDRV

- **G.FDRV** / **G.FDRV value**

Get or set the scaled value of the last pressed fader. This is the same as `G.FDR.V G.FDRI`. See G.FDR.V for more details.

G.FDRL

- **G.FDRL / G.FDRL level**

Get or set the brightness level (for coarse faders), or the maximum value level (for fine faders) of the last pressed fader. This is the same as `G.FDR.L G.BTNI`. For more details on levels see `G.FDR`.

G.FDR.PR

- **G.FDR.PR id value**

Emulate pressing the specified fader. Fader value will be set to the specified value, and if there is a script assigned it will be executed.

G.GFDR.N

- **G.GFDR.N group value**

Set the value for all faders in the specified group. This can be useful for resetting all faders in a group. See `G.FDR.N` for more details.

G.GFDR.V

- **G.GFDR.V group value**

Set the scaled value for all faders in the specified group. This can be useful for resetting all faders in a group. See `G.FDR.V` for more details.

G.GFDR.L

- **G.GFDR.L group odd_level even_level**

Set the brightness level (0..15) for all faders in the specified group. You can use different values for odd and even faders (based on their index within the group, not their id) - this can be a good way to provide some visual guidance.

G.GFDR.RN

- **G.GFDR.RN group min max**

Set the range to be used for V fader values (`G.FDR.V`, `G.FDRV`, `G.GFDR.V`). While the `.N` ops provide the actual fader value sometimes it's more convenient to map it to a different range so it can be used directly for something like a CV without having to scale it each time.

An example: let's say you create a coarse fader with the width of 8 which will be used to control a CV output where the voltage must be in the 2V..5V range. Using `G.FDR.N` you would need to do this: `CV 1 SCL 0 7 V 2 V 5 G.FDR.N 0`. Instead you can

set the range for scaling once: `G.GFDR.RN 0 V 2 V 5` (assuming the fader is in group 0) and then simply do `CV 1 G.FDR.V 0`.

The range is shared by all faders within the same group. If you need to use a different range use a different group when initializing a fader.

The default range is 0..16383. `G.RST` and `G.GRP.RST` reset ranges to the default value.

MIDI in

MIDI in ops allow the Teletype to react to MIDI events. MIDI is received via the USB port - simply plug a MIDI controller or sequencer into the USB port. Unless your MIDI device is powered externally, make sure your power supply can provide sufficient power! Please note that not all devices are supported.

To use the MIDI in ops, you need to assign MIDI events to one of the scripts with `MI.$` op. You can assign different event types to different scripts (so, script 1 could react to Note On events and script 2 to Note Off, for instance). You can assign multiple event types to the same script too. Various ops allow you to get detailed information about the event type and any additional data. It's possible that more than one event happens before a script is called (say, if you turn multiple knobs at once or play chords). To properly process them all, use indexed ops to get each event data instead of only processing the last event. The indexed ops use variable `I` as the index to allow easy use in loops.

OP	OP *(set)*	*(aliases)*	Description
`MI.$ x`	**`MI.$ x y`**		assign MIDI event type x to script y
`MI.LE`			get the latest event type
`MI.LCH`			get the latest channel (1..16)
`MI.LN`			get the latest Note On (0..127)
`MI.LNV`			get the latest Note On scaled to teletype range (shortcut for `N MI.LN`)
`MI.LV`			get the latest velocity (0..127)
`MI.LVV`			get the latest velocity scaled to 0..16383 range (0..+10V)
`MI.LO`			get the latest Note Off (0..127)
`MI.LC`			get the latest controller number (0..127)
`MI.LCC`			get the latest controller value (0..127)
`MI.LCCV`			get the latest controller value scaled to 0..16383 range (0..+10V)
`MI.NL`			get the number of Note On events
`MI.NCH`			get the Note On event channel (1..16) at index specified by variable `I`
`MI.N`			get the Note On (0..127) at index specified by variable `I`

OP	OP *(set)*	*(aliases)*	Description
MI.NV			get the Note On scaled to 0..+10V range at index specified by variable I
MI.V			get the velocity (0..127) at index specified by variable I
MI.VV			get the velocity scaled to 0..10V range at index specified by variable I
MI.OL			get the number of Note Off events
MI.OCH			get the Note Off event channel (1..16) at index specified by variable I
MI.O			get the Note Off (0..127) at index specified by variable I
MI.CL			get the number of controller events
MI.CCH			get the controller event channel (1..16) at index specified by variable I
MI.C			get the controller number (0..127) at index specified by variable I
MI.CC			get the controller value (0..127) at index specified by variable I
MI.CCV			get the controller value scaled to 0..+10V range at index specified by variable I
MI.CLKD	**MI.CLKD x**		set clock divider to x (1-24) or get the current divider
MI.CLKR			reset clock counter

MI.$

- **MI.$ x** / **MI.$ x y**

Assign a script to be triggered when a MIDI event of the specified type is received. The following types are supported: 0 - all events 1 - note on 2 - note off 3 - controller change 4 - clock 5 - start 6 - stop 7 - continue

Calibration

OP	OP *(set)*	*(aliases)*	Description
DEVICE.FLIP			Flip the screen/inputs/outputs
IN.CAL.MIN			Reads the input CV and assigns the voltage to the zero point
IN.CAL.MAX			Reads the input CV and assigns the voltage to the max point
IN.CAL.RESET			Resets the input CV calibration
PARAM.CAL.MIN			Reads the Parameter Knob minimum position and assigns a zero value
PARAM.CAL.MAX			Reads the Parameter Knob maximum position and assigns the maximum point
PARAM.CAL.RESET			Resets the Parameter Knob calibration
CV.CAL n mv1v mv3v			Calibrate CV output n
CV.CAL.RESET n			Reset calibration data for CV output n

DEVICE.FLIP

- **DEVICE.FLIP**

Flip the screen, the inputs and the outputs. This op is useful if you want to mount your Teletype upside down. The new state will be saved to flash.

IN.CAL.MIN

- **IN.CAL.MIN**

1. Connect a patch cable from a calibrated voltage source
2. Set the voltage source to 0 volts
3. Execute IN.CAL.MIN from the live terminal
4. Call IN and confirm the 0 result

IN.CAL.MAX

- **IN.CAL.MAX**

5. Set the voltage source to target maximum voltage (10V)
6. Execute IN.CAL.MAX from the live terminal
7. Call IN and confirm that the result is 16383

PARAM.CAL.MIN

- **PARAM.CAL.MIN**

    1. Turn the PARAM knob all the way to the left
    2. Execute PARAM.CAL.MIN from the live terminal
    3. Call PARAM and confirm the 0 result

PARAM.CAL.MAX

- **PARAM.CAL.MAX**

    4. Turn the knob all the way to the right
    5. Execute PARAM.CAL.MAX from the live terminal
    6. Call PARAM and verify that the result is 16383

CV.CAL

- **CV.CAL n mv1v mv3v**

Following a short calibration procedure, you can use `CV.CAL` to more precisely match your CV outputs to each other or to an external reference. A digital multimeter (or other voltage measuring device) is required.

To calibrate CV 1, first set it to output one volt with `CV 1 V 1`. Using a digital multimeter with at least millivolt precision (three digits after the decimal point), record the measured output of CV 1 between tip and sleeve on a patch cable. Then set CV 1 to three volts with `CV 1 V 3` and measure again.

Once you have both measurements, use the observed 1V and 3V values in millivolts as the second and third arguments to `CV.CAL`. For example, if you measured 0.990V and 2.984V, enter `CV.CAL 1 990 2984`. (If both your measurements are within 1 or 2 millivolts already, there's no need to run `CV.CAL`.)

Measure the output with `CV 1 V 1` and `CV 1 V 3` again and confirm the values are closer to the expected 1.000V and 3.000V.

Repeat the above steps for CV 2-4, if desired. The calibration data is stored in flash memory so you only need to go through this process once.

Note: The calibration adjustment is made after `CV.SLEW` and `CV.OFF` are applied, and does not affect `CV.GET` or any other scene-visible values. It only affects the levels coming out of the DAC.

CV.CAL.RESET

- **CV.CAL.RESET n**

Clear the calibration data for CV output n and return it to its default behavior, with no calibration adjustment.

Generic I2C

Generic I2C ops allow querying and sending commands to any I2C enabled devices connected to teletype. Before you can send or query you need to set the I2C address of the device using `II.A` (you might want to place that in your INIT script so that the address is set when you load a scene).

You can send up to 3 additional parameters, which can be either byte values or full range teletype values (for something like velocity), which will get sent as 2 bytes (MSB followed by LSB). All parameters must be of the same type - if you need to send both byte and word values, use the bitshift ops to combine/split bytes.

No validation or transformation is applied to any of the parameters - they are send as is. As dedicated ops are often 1-based, you might want to subtract 1 when reproducing them with the generic ops.

There are 2 sets of query ops - one for getting regular (word) values and one for getting byte values. If the address is not set, or if it's set but there are no follower devices listening at that address, query ops will return zero.

OP	OP *(set)*	*(aliases)*	Description
IIA	**IIA address**		Set I2C address or get the currently selected address
IIS cmd			Execute the specified command
IIS1 cmd value			Execute the specified command with 1 parameter
IIS2 cmd value1 value2			Execute the specified command with 2 parameters
IIS3 cmd value1 value2 value3			Execute the specified command with 3 parameters
IISB1 cmd value			Execute the specified command with 1 byte parameter
IISB2 cmd value1 value2			Execute the specified command with 2 byte parameters
IISB3 cmd value1 value2 value3			Execute the specified command with 3 byte parameters
IIQ cmd			Execute the specified query and get a value back
IIQ1 cmd value			Execute the specified query with 1 parameter and get a value back
IIQ2 cmd value1 value2			Execute the specified query with 2 parameters and get a value back
IIQ3 cmd value1 value2 value3			Execute the specified query with 3 parameters and get a value back

OP	OP *(set)*	*(aliases)*	Description
IIQB1 cmd value			Execute the specified query with 1 byte parameter and get a value back
IIQB2 cmd value1 value2			Execute the specified query with 2 byte parameters and get a value back
IIQB3 cmd value1 value2 value3			Execute the specified query with 3 byte parameters and get a value back
IIB cmd			Execute the specified query and get a byte value back
IIB1 cmd value			Execute the specified query with 1 parameter and get a byte value back
IIB2 cmd value1 value2			Execute the specified query with 2 parameters and get a byte value back
IIB3 cmd value1 value2 value3			Execute the specified query with 3 parameters and get a byte value back
IIBB1 cmd value			Execute the specified query with 1 byte parameter and get a byte value back
IIBB2 cmd value1 value2			Execute the specified query with 2 byte parameters and get a byte value back
IIBB3 cmd value1 value2 value3			Execute the specified query with 3 byte parameters and get a byte value back

IIA

- **IIA** / **IIA address**

Set the I2C address to be used by the generic I2C ops. The address is -1 when not selected or when it's set to a value outside of the supported range.

Ansible

OP	OP *(set)*	*(aliases)*	Description
ANS.G.LED x y			get grid LED buffer at position x, y
ANS.G x y	**ANS.G x y z**		get/set grid key on/off state (z) at position x, y
ANS.G.P x y			simulate grid key press at position (x, y)
ANS.A.LED n x			read arc LED buffer for ring n, LED x clockwise from north
ANS.A	**ANS.A n d**		send arc encoder event for ring n, delta d
ANS.APP	**ANS.APP x**		get/set active app
KR.PRE	**KR.PRE x**		return current preset / load preset x
KR.PERIOD	**KR.PERIOD x**		get/set internal clock period
KR.PAT	**KR.PAT x**		get/set current pattern
KR.SCALE	**KR.SCALE x**		get/set current scale
KR.POS x y	**KR.POS x y z**		get/set position z for track z, parameter y
KR.L.ST x y	**KR.L.ST x y z**		get loop start for track x, parameter y / set to z
KR.L.LEN x y	**KR.L.LEN x y z**		get length of track x, parameter y / set to z
KR.RES x y			reset position to loop start for track x, parameter y
KR.CV x			get the current CV value for channel x
KR.MUTE x	**KR.MUTE x y**		get/set mute state for channel x (1 = muted, 0 = unmuted)
KR.TMUTE x			toggle mute state for channel x
KR.CLK x			advance the clock for channel x (channel must have teletype clocking enabled)
KR.PG	**KR.PG x**		get/set the active page
KR.CUE	**KR.CUE x**		get/set the cued pattern
KR.DIR	**KR.DIR x**		get/set the step direction
KR.DUR x			get the current duration value for channel x
ME.PRE	**ME.PRE x**		return current preset / load preset x
ME.SCALE	**ME.SCALE x**		get/set current scale

OP	OP *(set)*	*(aliases)*	Description
ME.PERIOD	**ME.PERIOD x**		get/set internal clock period
ME.STOP x			stop channel x (0 = all)
ME.RES x			reset channel x (0 = all), also used as “start”
ME.CV x			get the current CV value for channel x
LV.PRE	**LV.PRE x**		return current preset / load preset x
LV.RES x			reset, 0 for soft reset (on next ext. clock), 1 for hard reset
LV.POS	**LV.POS x**		get/set current position
LV.L.ST	**LV.L.ST x**		get/set loop start
LV.L.LEN	**LV.L.LEN x**		get/set loop length
LV.L.DIR	**LV.L.DIR x**		get/set loop direction
LV.CV x			get the current CV value for channel x
CY.PRE	**CY.PRE x**		return current preset / load preset x
CY.RES x			reset channel x (0 = all)
CY.POS x	**CY.POS x y**		get / set position of channel x (x = 0 to set all), position between 0-255
CY.REV x			reverse channel x (0 = all)
CY.CV x			get the current CV value for channel x
MID.SLEW t			set pitch slew time in ms (applies to all allocation styles except FIXED)
MID.SHIFT o			shift pitch CV by standard Teletype pitch value (e.g. N 6, V -1, etc)
ARP.HLD h			0 disables key hold mode, other values enable
ARP.STY y			set base arp style [0-7]
ARP.GT v g			set voice gate length [0-127], scaled/synced to course divisions of voice clock
ARP.SLEW v t			set voice slew time in ms
ARP.RPT v n s			set voice pattern repeat, n times [0-8], shifted by s semitones [-24, 24]
ARP.DIV v d			set voice clock divisor (euclidean length), range [1-32]

OP	OP *(set)*	*(aliases)*	Description
`ARP.FIL v f`			set voice euclidean fill, use 1 for straight clock division, range [1-32]
`ARP.ROT v r`			set voice euclidean rotation, range [-32, 32]
`ARP.ER v f d r`			set all euclidean rhythm
`ARP.RES v`			reset voice clock/pattern on next base clock tick
`ARP.SHIFT v o`			shift voice cv by standard tt pitch value (e.g. N 6, V -1, etc)

ANS.APP

- **`ANS.APP`** / **`ANS.APP x`**

Get or change the app that is active on Ansible. Numbering:

- `0` = levels
- 1 = cycles
- 2 = kria
- 3 = meadowphysics
- 4 = midi standard
- 5 = midi arp
- 6 = teletype expander

KR.POS

- **`KR.POS x y`** / **`KR.POS x y z`**

Set position to z for track x, parameter y.

A value of `0` for x means all tracks.

A value of `0` for y means all parameters

Parameters:

- `0` = all
- 1 = trigger
- 2 = note
- 3 = octave
- 4 = length

KR.PG

- **`KR.PG`** / **`KR.PG x`**

Get or change the current parameter page. Numbering:

- 0 = trigger
- 1 = ratchet
- 2 = note
- 3 = alt note
- 4 = octave
- 5 = glide
- 6 = duration
- 7 = TBD
- 8 = scale
- 9 = pattern

KR.CUE

- **KR.CUE** / **KR.CUE x**

Get or change the cued pattern. Numbered from 0.

KR.DIR

- **KR.DIR** / **KR.DIR x**

Get or change the step direction. Numbered from 0.

White Whale

OP	OP *(set)*	*(aliases)*	Description
WW.PRESET x			Recall preset (0-7)
WW.POS x			Cut to position (0-15)
WW.SYNC x			Cut to position (0-15) and hard-sync the clock (if clocked internally)
WW.START x			Set the loop start position (0-15)
WW.END x			Set the loop end position (0-15)
WW.PMODE x			Set the loop play mode (0-5)
WW.PATTERN x			Change pattern (0-15)
WW.QPATTERN x			Change pattern (0-15) after current pattern ends
WW.MUTE1 x			Mute trigger 1 (0 = on, 1 = mute)
WW.MUTE2 x			Mute trigger 2 (0 = on, 1 = mute)
WW.MUTE3 x			Mute trigger 3 (0 = on, 1 = mute)
WW.MUTE4 x			Mute trigger 4 (0 = on, 1 = mute)
WW.MUTEA x			Mute CV A (0 = on, 1 = mute)
WW.MUTEB x			Mute CV B (0 = on, 1 = mute)

WW.PRESET

- **WW.PRESET x**

Set White Whale to preset x (0-7). This takes effect immediately. The current playback position is not changed.

WW.POS

- **WW.POS x**

Cut immediately to position (0-15) in the currently playing pattern.

WW.SYNC

- **WW.SYNC x**

Cut to position (0-15) in the currently playing pattern. If White Whale is being clocked internaly, this also hard-syncs the clock.

WW.START

- **WW.START x**

Set the loop start position (0-15). This does not impact the current playback position. If the playback position is outside of the defined loop it will continue to step until it

enters the loop. If the start position is after the end position, the loop will wrap around the ends of the grid.

WW.END

- **WW.END x**

Set the loop end position (0-15). This does not impact the current playback position. If the playback position is outside of the defined loop it will continue to step until it enters the loop. If the end position is before the end position, the loop will wrap around the ends of the grid.

WW.PMODE

- **WW.PMODE x**

Set the loop play mode. The available modes are: 0 - forward, 1 - reverse, 2 - drunk, 3 - random, 4 - pingpong, 5 - pingpong with repeated end points.

WW.PATTERN

- **WW.PATTERN x**

Change pattern. This does not impact the current playback position.

WW.QPATTERN

- **WW.QPATTERN x**

Change pattern (0-15) after current pattern ends

WW.MUTE1

- **WW.MUTE1 x**

Mute trigger 1 (0 = on, 1 = mute).

WW.MUTE2

- **WW.MUTE2 x**

Mute trigger 2 (0 = on, 1 = mute).

WW.MUTE3

- **WW.MUTE3 x**

Mute trigger 3 (0 = on, 1 = mute).

WW.MUTE4

- **WW.MUTE4 x**

Mute trigger 4 (0 = on, 1 = mute).

WW.MUTEA

- **WW.MUTEA x**

Mute CV A (0 = on, 1 = mute).

WW.MUTEB

- **WW.MUTEB x**

Mute CV B (0 = on, 1 = mute).

Meadowphysics

For use on the original Meadowphysics module with version 2 firmware. Reference the Ansible ops for using Meadowphysics on the Ansible module.

OP	OP *(set)*	*(aliases)*	Description
`MP.PRESET x`			set Meadowphysics to preset x (indexed from `0`)
`MP.RESET x`			reset countdown for channel x (`0` = all, `1`–`8` = individual channels)
`MP.STOP x`			reset channel x (`0` = all, `1`–`8` = individual channels)

Earthsea

OP	OP *(set)*	*(aliases)*	Description
ES.PRESET x			Recall preset x (0-7)
ES.MODE x			Set pattern clock mode. (0=normal, 1=II clock)
ES.CLOCK x			If II clocked, next pattern event
ES.RESET x			Reset pattern to start (and start playing)
ES.PATTERN x			Select playing pattern (0-15)
ES.TRANS x			Transpose the current pattern
ES.STOP x			Stop pattern playback.
ES.TRIPLE x			Recall triple shape (1-4)
ES.MAGIC x			Magic shape (1= halfspeed, 2=doublespeed, 3=linearize)
ES.CV x			get the current CV value for channel x

ES.PRESET

- **ES.PRESET x**

Recall the preset in location x. This will stop the currently playing pattern.

ES.MODE

- **ES.MODE x**

Sets the pattern clock mode. Setting x to 0 sets Earthsea to use it's internal clock. Setting x to 1 clocks Earthsea via the ES.CLOCK command.

ES.CLOCK

- **ES.CLOCK x**

If Earthsea is II clocked (see ES.MODE), and x is non-zero, advance to the next pattern event.

ES.RESET

- **ES.RESET x**

If x is non-zero, reset the position in the current pattern to the start and start playing.

ES.PATTERN

- **ES.PATTERN x**

Select pattern (0-15) from the current preset.

ES.TRANS

- **ES.TRANS x**

Apply a transposition relative to the current ‘root’ position. Integer divisions of x shift the root note up or down a row, x modulo 5 will shift the position left or right up to 4 notes.

ES.STOP

- **ES.STOP x**

If x is non-zero, stop pattern playback, or stop record if currently recording.

ES.TRIPLE

- **ES.TRIPLE x**

Recall triple shape (1-4).

ES.MAGIC

- **ES.MAGIC x**

Apply one of the magic shapes, (1= halfspeed, 2=doublespeed, 3=linearize). Other shapes are not currently available via II ops.

Orca

Remote commands for Orca (alternative WW firmware). For detailed info and tips on usage please refer to the Orca manual[29].

OP	OP *(set)*	*(aliases)*	Description
OR.CLK x			Advance track x (1-4)
OR.RST x			Reset track x (1-4)
OR.GRST x			Global reset (x can be any value)
OR.TRK x			Choose track x (1-4) to be used by OR.DIV, OR.PHASE, OR.WGT or OR.MUTE
OR.DIV x			Set divisor for selected track to x (1-16)
OR.PHASE x			Set phase for selected track to x (0-16)
OR.WGT x			Set weight for selected track to x (1-8)
OR.MUTE x			Mute trigger selected by OR.TRK (0 = on, 1 = mute)
OR.SCALE x			Select scale x (1-16)
OR.BANK x			Select preset bank x (1-8)
OR.PRESET x			Select preset x (1-8)
OR.RELOAD x			Reload preset or bank (0 - current preset, 1 - current bank, 2 - all banks)
OR.ROTS x			Rotate scales by x (1-15)
OR.ROTW x			Rotate weights by x (1-3)
OR.CVA x			Select tracks for CV A where x is a binary number representing the tracks
OR.CVB x			Select tracks for CV B where x is a binary number representing the tracks

OR.CLK

- **OR.CLK x**

Gives you the ability to clock individual tracks. The master clock will still advance all 4 tracks.

[29] https://github.com/scanner-darkly/monome-mods/wiki/Orca---manual#teletype-integration

OR.SCALE

- **OR.SCALE x**

Value of 1-16 will select scale for both CV A and CV B. To select individual scales append their numbers, for instance, 105 will select scale 1 for CV A and scale 5 for CV B, and 1005 will select scale 10 for CV A and scale 5 for CV B.

OR.RELOAD

- **OR.RELOAD x**

Abandons any unsaved changes and reloads selected presets/banks from flash. Could be useful in I script.

OR.ROTS

- **OR.ROTS x**

Rotates scales up. To rotate them down set x to 16 minus the amount.

OR.ROTW

- **OR.ROTW x**

Rotates weights up. To rotate them down set x to 4 minus the amount.

OR.CVA

- **OR.CVA x**

Convert a binary number representing selected tracks (so 1001 will select tracks 1 and 4, for instance) and set x to that.

OR.CVB

- **OR.CVB x**

Convert a binary number representing selected tracks (so 1001 will select tracks 1 and 4, for instance) and set x to that.

Just Friends

More extensively covered in the Just Friends Documentation[30].

OP	OP *(set)*	*(aliases)*	Description
`JF.ADDR x`			Sets JF II address (1 = primary, 2 = secondary). Use with only one JF on the bus! Saves to JF internal memory, so only one-time config is needed.
`JF.SEL x`			Sets target JF unit (1 = primary, 2 = secondary).
`JF0: ...`			Send following JF OPs to both units starting with selected unit.
`JF1: ...`			Send following JF OPs to unit 1 ignoring the currently selected unit.
`JF2: ...`			Send following JF OPs to unit 2 ignoring the currently selected unit.
`JF.RAMP`			Gets value of RAMP knob.
`JF.CURVE`			Gets value of CURVE knob.
`JF.FM`			Gets value of FM knob.
`JF.INTONE`			Gets value of INTONE knob and CV offset.
`JF.TIME`			Gets value of TIME knob and CV offset.
`JF.SPEED`			Gets value of SPEED switch (1 = sound, `0` = shape).
`JF.TSC`			Gets value of MODE switch (`0` = transient, 1 = sustain, 2 = cycle).
`JF.TR x y`			Simulate a TRIGGER input. `x` is channel (`0` = all primary JF channels, `1..6` = primary JF, `7..12` = secondary JF, `-1` = all channels both JF) and y is state (`0` or 1)
`JF.RMODE x`			Set the RUN state of Just Friends when no physical jack is present. (`0` = run off, non-zero = run on)

[30] https://github.com/whimsicalraps/Just-Friends/blob/main/Just-Type.md

OP	OP *(set)*	*(aliases)*	Description
JF.RUN x			Send a 'voltage' to the RUN input. Requires JF.RMODE 1 to have been executed, or a physical cable in JF's input. Thus Just Friend's RUN modes are accessible without needing a physical cable & control voltage to set the RUN parameter. use JF.RUN V x to set to x volts. The expected range is V -5 to V 5
JF.SHIFT x			Shifts the transposition of Just Friends, regardless of speed setting. Shifting by V 1 doubles the frequency in sound, or doubles the rate in shape. x = pitch, use N x for semitones, or V y for octaves.
JF.VTR x y			Like JF.TR with added volume control. Velocity is scaled with volts, so try V 5 for an output trigger of 5 volts. Channels remember their latest velocity setting and apply it regardless of TRIGGER origin (digital or physical). x = channel, 0 sets all channels. y = velocity, amplitude of output in volts. eg JF.VTR 1 V 4.
JF.TUNE x y z			Adjust the tuning ratios used by the INTONE control. x = channel, y = numerator (set the multiplier for the tuning ratio), z = denominator (set the divisor for the tuning ratio). JF.TUNE 0 0 0 resets to default ratios.
JF.MODE x			Set the current choice of standard functionality, or Just Type alternate modes (Speed switch to Sound for Synth, Shape for Geode). You'll likely want to put JF.MODE x in your Teletype INIT scripts. x = nonzero activates alternative modes. 0 restores normal.

OP	OP *(set)*	*(aliases)*	Description
JF.VOX x y z			Synth mode: create a note at the specified channel, of the defined pitch & velocity. All channels can be set simultaneously with a chan value of 0. x = channel, y = pitch relative to C3, z = velocity (like JF.VTR). Geode mode: Create a stream of rhythmic envelopes on the named channel. x = channel, y = division, z = number of repeats.
JF.NOTE x y			Synth: polyphonically allocated note sequencing. Works as JF.VOX with chan selected automatically. Free voices will be taken first. If all voices are busy, will steal from the voice which has been active the longest. x = pitch relative to C3, y = velocity. Geode: works as JF.VOX with dynamic allocation of channel. Assigns the rhythmic stream to the oldest unused channel, or if all are busy, the longest running channel. x = division, y = number of repeats.
JF.POLY x y			As JF.NOTE but across dual JF. Switches between primary and secondary units every 6 notes or until reset using JF.POLY.RESET.
JF.POLY.RESET			Resets JF.POLY note count.
JF.PITCH x y			Change pitch without retriggering. x = channel, y = pitch relative to C3.
JF.GOD x			Redefines C3 to align with the 'God' note. x = 0 sets A to 440, x = 1 sets A to 432.
JF.TICK x			Sets the underlying timebase of the Geode. x = clock. 0 resets the timebase to the start of measure. 1 to 48 shall be sent repetitively. The value representing ticks per measure. 49 to 255 sets beats-per-minute and resets the timebase to start of measure.

OP	OP *(set)*	*(aliases)*	Description
JF.QT x			When non-zero, all events are queued & delayed until the next quantize event occurs. Using values that don't align with the division of rhythmic streams will cause irregular patterns to unfold. Set to 0 to deactivate quantization. x = division, 0 deactivates quantization, 1 to 32 sets the subdivision & activates quantization.

16n

The 16n Faderbank is an open-source controller that can be polled by the Teletype to read the positions of its 16 sliders.

OP	OP *(set)*	*(aliases)*	Description
FADER x		**FB**	Reads the value of the FADER slider x; default return range is from 0 to 16383. Up to four Faderbanks can be addressed; x value between 1 and 16 correspond to Faderbank 1, x between 17 and 32 to Faderbank 2, etc…
FADER.SCALE x y z		**FB.S**	Set static scaling of the FADER x to between `min` and `max`.
FADER.CAL.MIN x		**FB.C.MIN**	Reads FADER x minimum position and assigns a zero value
FADER.CAL.MAX x		**FB.C.MAX**	Reads FADER x maximum position and assigns the maximum point
FADER.CAL.RESET x		**FB.C.R**	Resets the calibration for FADER x

FADER.CAL.MIN

- **FADER.CAL.MIN x**
- *alias*: **FB.C.MIN**

1. Slide FADER x all the way down to the bottom
2. Execute FADER.CAL.MIN x from the live terminal
3. Call FADER x and confirm the 0 result

FADER.CAL.MAX

- **FADER.CAL.MAX x**
- *alias*: **FB.C.MAX**

1. Slide FADER x all the way up to the top
2. Execute FADER.CAL.MAX x from the live terminal
3. Call FADER x and verify that the result is 16383

ER-301

The Orthogonal Devices ER-301 Sound Computer is a voltage-controllable canvas for digital signal processing algorithms available from Orthogonal Devices. It can communicate with the Teletype to send up to 100 triggers and 100 CV values per device. Up to three devices are software-selectable and correlate to outputs up to 300.

OP	OP *(set)*	*(aliases)*	Description
SC.TR x y			Set trigger output for the ER-301 virtual output x to y (0-1)
SC.TR.POL x y			Set polarity of trigger for the ER-301 virtual output x to y (0-1)
SC.TR.TIME x y			Set the pulse time for the ER-301 virtual trigger x to y in ms
SC.TR.TOG x			Flip the state for the ER-301 virtual trigger output x
SC.TR.PULSE x		**SC.TR.P**	Pulse the ER-301 virtual trigger output x
SC.CV x y			CV target value for the ER-301 virtual output x to value y
SC.CV.OFF x y			CV offset added to the ER-301 virtual output x
SC.CV.SET x			Set CV value for the ER-301 virtual output x
SC.CV.SLEW x y			Set the CV slew time for the ER-301 virtual output x in ms

TELEXi

The TELEXi (or TXi) is an input expander that adds 4 IN jacks and 4 PARAM knobs to the Teletype. There are jumpers on the back so you can hook more than one TXi to your Teletype simultaneously.

Inputs added to the system by the TELEX modules are addressed sequentially: 1-4 are on your first module of any type, 5-8 are on the second, 9-12 on the third, and so on. A few of the commands reference the module by its unit number – but those are rare.

OP	OP *(set)*	*(aliases)*	Description
TI.PARAM x		**TI.PRM**	reads the value of PARAM knob x; default return range is from 0 to 16383; return range can be altered by the TI.PARAM.MAP command
TI.PARAM.QT x		**TI.PRM.QT**	return the quantized value for PARAM knob x using the scale set by TI.PARAM.SCALE; default return range is from 0 to 16383
TI.PARAM.N x		**TI.PRM.N**	return the quantized note number for PARAM knob x using the scale set by TI.PARAM.SCALE
TI.PARAM.SCALE x		**TI.PRM.SCALE**	select scale # y for PARAM knob x; scales listed in full description
TI.PARAM.MAP x y z		**TI.PRM.MAP**	maps the PARAM values for input x across the range y - z (defaults 0-16383)
TI.IN x			reads the value of IN jack x; default return range is from -16384 to 16383 - representing -10V to +10V; return range can be altered by the TI.IN.MAP command
TI.IN.QT x			return the quantized value for IN jack x using the scale set by TI.IN.SCALE; default return range is from -16384 to 16383 - representing -10V to +10V
TI.IN.N x			return the quantized note number for IN jack x using the scale set by TI.IN.SCALE
TI.IN.SCALE x			select scale # y for IN jack x; scales listed in full description

OP	OP *(set)*	*(aliases)*	Description
`TI.IN.MAP x y z`			maps the IN values for input jack x across the range y - z (default range is -16384 to 16383 - representing -10V to +10V)
`TI.PARAM.INIT x`		**`TI.PRM.INIT`**	initializes PARAM knob x back to the default boot settings and behaviors; neutralizes mapping (but not calibration)
`TI.IN.INIT x`			initializes IN jack x back to the default boot settings and behaviors; neutralizes mapping (but not calibration)
`TI.INIT d`			initializes all of the PARAM and IN inputs for device number d (1-8)
`TI.PARAM.CALIB x y`		**`TI.PRM.CALIB`**	calibrates the scaling for PARAM knob x; y of 0 sets the bottom bound; y of 1 sets the top bound
`TI.IN.CALIB x y`			calibrates the scaling for IN jack x; y of -1 sets the -10V point; y of 0 sets the 0V point; y of 1 sets the +10V point
`TI.STORE d`			stores the calibration data for TXi number d (1-8) to its internal flash memory
`TI.RESET d`			resets the calibration data for TXi number d (1-8) to its factory defaults (no calibration)

TI.PARAM.SCALE

- **`TI.PARAM.SCALE x`**
- *alias*: **`TI.PRM.SCALE`**

Quantization Scales

0. Equal Temperament [DEFAULT]
1. 12-tone Pythagorean scale
2. Vallotti & Young scale (Vallotti version) also known as Tartini-Vallotti (1754)
3. Andreas Werckmeister's temperament III (the most famous one, 1681)
4. Wendy Carlos' Alpha scale with perfect fifth divided in nine
5. Wendy Carlos' Beta scale with perfect fifth divided by eleven
6. Wendy Carlos' Gamma scale with third divided by eleven or fifth by twenty
7. Carlos Harmonic & Ben Johnston's scale of 'Blues' from Suite f.micr.piano (1977) & David Beardsley's scale of 'Science Friction'

8. Carlos Super Just
9. Kurzweil “Empirical Arabic”
10. Kurzweil “Just with natural b7th”, is Sauveur Just with 7/4
11. Kurzweil “Empirical Bali/Java Harmonic Pelog”
12. Kurzweil “Empirical Bali/Java Slendro, Siam 7”
13. Kurzweil “Empirical Tibetian Ceremonial”
14. Harry Partch’s 43-tone pure scale
15. Partch’s Indian Chromatic, Exposition of Monophony, 1933.
16. Partch Greek scales from “Two Studies on Ancient Greek Scales” on black/white

TI.PARAM.MAP

- **`TI.PARAM.MAP x y z`**
- *alias*: **`TI.PRM.MAP`**

If you would like to have a `PARAM` knob values over a specific range, you can offload the processing for this to the TXo by mapping the range of the potentiometer using the `MAP` command. It works a lot like the `MAP` operator, but does the heavy lifting on the TXi, saving you space in your code and cycles on your processor.

For instance, let’s have the first knob return a range from 0 to 100.

```
TI.PARAM.MAP 1 0 100
```

You can reset the mapping by either calling the map command with the default range or by using the `INIT` command (`TO.PARAM.INIT 1`).

TI.IN.SCALE

- **`TI.IN.SCALE x`**

Quantization Scales

0. Equal Temperament [DEFAULT]
1. 12-tone Pythagorean scale
2. Vallotti & Young scale (Vallotti version) also known as Tartini-Vallotti (1754)
3. Andreas Werckmeister’s temperament III (the most famous one, 1681)
4. Wendy Carlos’ Alpha scale with perfect fifth divided in nine
5. Wendy Carlos’ Beta scale with perfect fifth divided by eleven
6. Wendy Carlos’ Gamma scale with third divided by eleven or fifth by twenty
7. Carlos Harmonic & Ben Johnston’s scale of ‘Blues’ from Suite f.micr.piano (1977) & David Beardsley’s scale of ‘Science Friction’
8. Carlos Super Just
9. Kurzweil “Empirical Arabic”
10. Kurzweil “Just with natural b7th”, is Sauveur Just with 7/4
11. Kurzweil “Empirical Bali/Java Harmonic Pelog”
12. Kurzweil “Empirical Bali/Java Slendro, Siam 7”
13. Kurzweil “Empirical Tibetian Ceremonial”
14. Harry Partch’s 43-tone pure scale

15. Partch's Indian Chromatic, Exposition of Monophony, 1933.
16. Partch Greek scales from "Two Studies on Ancient Greek Scales" on black/white

TI.PARAM.CALIB

- **TI.PARAM.CALIB x y**
- *alias*: **TI.PRM.CALIB**

You can calibrate your PARAM knob by using this command. The steps for full calibration are as follows:

1. Turn the PARAM knob x all the way to the left
2. Send the command 'TI.PARAM.CALIBRATE x 0'
3. Turn the PARAM knob x all the way to the right
4. Send the command 'TI.PARAM.CALIBRATE x 1'

Don't forget to call the TI.STORE command to save your calibration between sessions.

TI.IN.CALIB

- **TI.IN.CALIB x y**

You can calibrate your IN jack to external voltages by using this command. The steps for full calibration are as follows:

1. Send a -10V signal to the input x
2. Send the command 'TI.IN.CALIBRATE x -1'
3. Send a 0V signal to the input x
4. Send the command 'TI.IN.CALIBRATE x 0'
5. Send a 10V signal to the input x
6. Send the command 'TI.IN.CALIBRATE x 1'

Don't forget to call the TI.STORE command to save your calibration between sessions.

TELEXo

The TELEXo (or TXo) is an output expander that adds an additional 4 Trigger and 4 CV jacks to the Teletype. There are jumpers on the back so you can hook more than one TXo to your Teletype simultaneously.

Outputs added to the system by the TELEX modules are addressed sequentially: 1-4 are on your first module of any type, 5-8 are on the second, 9-12 on the third, and so on. A few of the commands reference the module by its unit number – but those are rare.

Unlike the Teletype's equivalent operators, the TXo does not have get commands for its functions. This was intentional as these commands eat up processor and bus-space. While they may be added in the future, as of now you cannot poll the TXo for the current state of its various operators.

OP	OP *(set)*	*(aliases)*	Description
`TO.TR x y`			sets the TR value for output x to y (0/1)
`TO.TR.TOG x`			toggles the TR value for output x
`TO.TR.PULSE x`		**`TO.TR.P`**	pulses the TR value for output x for the duration set by `TO.TR.TIME/S/M`
`TO.TR.PULSE.DIV x y`		**`TO.TR.P.DIV`**	sets the clock division factor for TR output x to y
`TO.TR.PULSE.MUTE x y`		**`TO.TR.P.MUTE`**	mutes or un-mutes TR output x; y is 1 (mute) or 0 (un-mute)
`TO.TR.TIME x y`			sets the time for `TR.PULSE` on output n; y in milliseconds
`TO.TR.TIME.S x y`			sets the time for `TR.PULSE` on output n; y in seconds
`TO.TR.TIME.M x y`			sets the time for `TR.PULSE` on output n; y in minutes
`TO.TR.WIDTH x y`			sets the time for `TR.PULSE` on output n based on the width of its current metronomic value; y in percentage (0-100)
`TO.TR.POL x y`			sets the polarity for TR output n
`TO.TR.M.ACT x y`			sets the active status for the independent metronome for output x to y (`0`/1); default `0` (disabled)
`TO.TR.M x y`			sets the independent metronome interval for output x to y in milliseconds; default `1000`

OP	OP *(set)*	*(aliases)*	Description
TO.TR.M.S x y			sets the independent metronome interval for output x to y in seconds; default 1
TO.TR.M.M x y			sets the independent metronome interval for output x to y in minutes
TO.TR.M.BPM x y			sets the independent metronome interval for output x to y in Beats Per Minute
TO.TR.M.COUNT x y			sets the number of repeats before deactivating for output x to y; default `0` (infinity)
TO.TR.M.MUL x y			multiplies the `M` rate on TR output x by y; y defaults to 1 - no multiplication
TO.TR.M.SYNC x			synchronizes the PULSE for metronome on TR output number x
TO.M.ACT d y			sets the active status for the 4 independent metronomes on device d (1-8) to y (`0`/1); default `0` (disabled)
TO.M d y			sets the 4 independent metronome intervals for device d (1-8) to y in milliseconds; default `1000`
TO.M.S d y			sets the 4 independent metronome intervals for device d to y in seconds; default 1
TO.M.M d y			sets the 4 independent metronome intervals for device d to y in minutes
TO.M.BPM d y			sets the 4 independent metronome intervals for device d to y in Beats Per Minute
TO.M.COUNT d y			sets the number of repeats before deactivating for the 4 metronomes on device d to y; default `0` (infinity)
TO.M.SYNC d			synchronizes the 4 metronomes for device number d (1-8)
TO.CV x			CV target output x; y values are bipolar (-16384 to +16383) and map to -10 to +10
TO.CV.SLEW x y			set the slew amount for output x; y in milliseconds

OP	OP *(set)*	*(aliases)*	Description
TO.CV.SLEW.S x y			set the slew amount for output x; y in seconds
TO.CV.SLEW.M x y			set the slew amount for output x; y in minutes
TO.CV.SET x y			set the CV for output x (ignoring SLEW); y values are bipolar (-16384 to +16383) and map to -10 to +10
TO.CV.OFF x y			set the CV offset for output x; y values are added at the final stage
TO.CV.QT x y			CV target output x; y is quantized to output's current CV.SCALE
TO.CV.QT.SET x y			set the CV for output x (ignoring SLEW); y is quantized to output's current CV.SCALE
TO.CV.N x y			target the CV to note y for output x; y is indexed in the output's current CV.SCALE
TO.CV.N.SET x y			set the CV to note y for output x; y is indexed in the output's current CV.SCALE (ignoring SLEW)
TO.CV.SCALE x y			select scale # y for CV output x; scales listed in full description
TO.CV.LOG x y			translates the output for CV output x to logarithmic mode y; y defaults to 0 (off); mode 1 is for 0-16384 (0V-10V), mode 2 is for 0-8192 (0V-5V), mode 3 is for 0-4096 (0V-2.5V), etc.
TO.CV.CALIB x			Locks the current offset (CV.OFF) as a calibration offset and saves it to persist between power cycles for output x.
TO.CV.RESET x			Clears the calibration offset for output x
TO.OSC x y			Targets oscillation for CV output x to y
TO.OSC.SET x y			set oscillation for CV output x to y (ignores slew)
TO.OSC.QT x y			targets oscillation for CV output x to y

OP	OP *(set)*	*(aliases)*	Description
TO.OSC.QT.SET x y			set oscillation for CV output x to y, quantized to the current scale (ignores slew)
TO.OSC.N x y			targets oscillation for CV output x to note y
TO.OSC.N.SET x y			sets oscillation for CV output x to note y (ignores slew)
TO.OSC.FQ x y			targets oscillation for CV output x to frequency y in Hertz
TO.OSC.FQ.SET x y			targets oscillation for CV output x to frequency y in Hertz (ignores slew)
TO.OSC.LFO x y			Targets oscillation for CV output x to LFO frequency y in millihertz
TO.OSC.LFO.SET x y			Targets oscillation for CV output x to LFO frequency y in millihertz (ignores slew)
TO.OSC.CYC x y			targets the oscillator cycle length to y for CV output x with the portamento rate determined by the `TO.OSC.SLEW` value; y is in milliseconds
TO.OSC.CYC.SET x y			sets the oscillator cycle length to y for CV output x (ignores `CV.OSC.SLEW`); y is in milliseconds
TO.OSC.CYC.S x y			targets the oscillator cycle length to y for CV output x with the portamento rate determined by the `TO.OSC.SLEW` value; y is in seconds
TO.OSC.CYC.S.SET x y			sets the oscillator cycle length to y for CV output x (ignores `CV.OSC.SLEW`); y is in seconds
TO.OSC.CYC.M x y			targets the oscillator cycle length to y for CV output x with the portamento rate determined by the `TO.OSC.SLEW` value; y is in minutes
TO.OSC.CYC.M.SET x y			sets the oscillator cycle length to y for CV output x (ignores `CV.OSC.SLEW`); y is in minutes
TO.OSC.SCALE x y			select scale # y for CV output x; scales listed in full description

OP	OP *(set)*	*(aliases)*	Description
TO.OSC.WAVE x y			set the waveform for output x to y; y range is 0–4500, blending between 45 waveforms
TO.OSC.RECT x y			rectifies the polarity of the oscillator for output x to y; 0 is no rectification, +/-1 is partial rectification, +/-2 is full rectification
TO.OSC.WIDTH x y			sets the width of the pulse wave on output x to y; y is a percentage of total width (0 to 100); only affects waveform 3000
TO.OSC.SYNC x			resets the phase of the oscillator on CV output x (relative to TO.OSC.PHASE)
TO.OSC.PHASE x y			sets the phase offset of the oscillator on CV output x to y (0 to 16383); y is the range of one cycle
TO.OSC.SLEW x y			sets the frequency slew time (portamento) for the oscillator on CV output x to y; y in milliseconds
TO.OSC.SLEW.S x y			sets the frequency slew time (portamento) for the oscillator on CV output x to y; y in seconds
TO.OSC.SLEW.M x y			sets the frequency slew time (portamento) for the oscillator on CV output x to y; y in minutes
TO.OSC.CTR x y			centers the oscillation on CV output x to y; y values are bipolar (-16384 to +16383) and map to -10 to +10
TO.ENV.ACT x y			activates/deactivates the AD envelope generator for the CV output x; y turns the envelope generator off (0 - default) or on (1); CV amplitude is used as the peak for the envelope and needs to be > 0 for the envelope to be perceivable
TO.ENV x y			trigger the attack stage of output x when y changes to 1, or decay stage when it changes to 0

OP	OP *(set)*	*(aliases)*	Description
TO.ENV.TRIG x			triggers the envelope at CV output x to cycle; CV amplitude is used as the peak for the envelope and needs to be > 0 for the envelope to be perceivable
TO.ENV.ATT x y			set the envelope attack time to y for CV output x; y in milliseconds (default 12 ms)
TO.ENV.ATT.S x y			set the envelope attack time to y for CV output x; y in seconds
TO.ENV.ATT.M x y			set the envelope attack time to y for CV output x; y in minutes
TO.ENV.DEC x y			set the envelope decay time to y for CV output x; y in milliseconds (default 250 ms)
TO.ENV.DEC.S x y			set the envelope decay time to y for CV output x; y in seconds
TO.ENV.DEC.M x y			set the envelope decay time to y for CV output x; y in minutes
TO.ENV.EOR x n			at the end of rise of CV output x, fires a PULSE to the trigget output n
TO.ENV.EOC x n			at the end of cycle of CV output x, fires a PULSE to the trigget output n
TO.ENV.LOOP x y			causes the envelope on CV output x to loop for y times
TO.TR.INIT x			initializes TR output x back to the default boot settings and behaviors; neutralizes metronomes, dividers, pulse counters, etc.
TO.CV.INIT x			initializes CV output x back to the default boot settings and behaviors; neutralizes offsets, slews, envelopes, oscillation, etc.
TO.INIT d			initializes all of the TR and CV outputs for device number d (1-8)
TO.KILL d			cancels all TR pulses and CV slews for device number d (1-8)

TO.TR.PULSE.DIV

- **TO.TR.PULSE.DIV x y**
- *alias*: **TO.TR.P.DIV**

The pulse divider will output one trigger pulse every y pulse commands. For example, setting the DIV factor to 2 like this:

```
TO.TR.P.DIV 1 2
```

Will cause every other `TO.TR.P 1` command to emit a pulse.

Reset it to one (`TO.TR.P.DIV 1 1`) or initialize the output (`TO.TR.INIT 1`) to return to the default behavior.

TO.TR.WIDTH

- **TO.TR.WIDTH x y**

The actual time value for the trigger pulse when set by the WIDTH command is relative to the current value for `TO.TR.M`. Changes to `TO.TR.M` will change the duration of TR.PULSE when using the WIDTH mode to set its value. Values for y are set in percentage (0-100).

For example:

```
TO.TR.M 1 1000
TO.TR.WIDTH 1 50
```

The length of a TR.PULSE is now 500ms.

```
TO.TR.M 1 500
```

The length of a TR.PULSE is now 250ms. Note that you don't need to use the width command again as it automatically tracks the value for `TO.TR.M`.

TO.TR.M.ACT

- **TO.TR.M.ACT x y**

Each TR output has its own independent metronome that will execute a TR.PULSE at a specified interval. The ACT command enables (1) or disables (0) the metronome.

TO.TR.M.COUNT

- **TO.TR.M.COUNT x y**

This allows for setting a limit to the number of times `TO.TR.M` will PULSE when active before automatically disabling itself. For example, let's set it to pulse 5 times with 500ms between pulses:

```
TO.TR.M 1 500
TO.TR.M.COUNT 1 5
```

Now, each time we activate it, the metronome will pulse 5 times - each a half-second apart.

```
TO.TR.M.ACT 1 1
```

PULSE … PULSE … PULSE … PULSE … PULSE.

The metronome is now disabled after pulsing five times. If you call ACT again, it will emit five more pulses.

To reset, either set your COUNT to zero (`TO.TR.M.COUNT 1 0`) or call init on the output (`TO.TR.INIT 1 1`).

TO.TR.M.MUL

- **TO.TR.M.MUL x y**

The following example will cause 2 against 3 patterns to pulse out of TO.TR outputs 3 and 4.

```
TO.TR.M.MUL 3 2
TO.TR.M.MUL 4 3
L 3 4: TO.TR.M.ACT I 1
```

TO.M.SYNC

- **TO.M.SYNC d**

This command causes the TXo at device d to synchronize all of its independent metronomes to the moment it receives the command. Each will then continue to pulse at its own independent M rate.

TO.CV.SCALE

- **TO.CV.SCALE x y**

Quantization Scales

0. Equal Temperament [DEFAULT]
1. 12-tone Pythagorean scale
2. Vallotti & Young scale (Vallotti version) also known as Tartini-Vallotti (1754)
3. Andreas Werckmeister's temperament III (the most famous one, 1681)
4. Wendy Carlos' Alpha scale with perfect fifth divided in nine
5. Wendy Carlos' Beta scale with perfect fifth divided by eleven
6. Wendy Carlos' Gamma scale with third divided by eleven or fifth by twenty
7. Carlos Harmonic & Ben Johnston's scale of 'Blues' from Suite f.micr.piano (1977) & David Beardsley's scale of 'Science Friction'
8. Carlos Super Just
9. Kurzweil "Empirical Arabic"
10. Kurzweil "Just with natural b7th", is Sauveur Just with 7/4

11. Kurzweil “Empirical Bali/Java Harmonic Pelog”
12. Kurzweil “Empirical Bali/Java Slendro, Siam 7”
13. Kurzweil “Empirical Tibetian Ceremonial”
14. Harry Partch’s 43-tone pure scale
15. Partch’s Indian Chromatic, Exposition of Monophony, 1933.
16. Partch Greek scales from “Two Studies on Ancient Greek Scales” on black/white

TO.CV.LOG

- **TO.CV.LOG x y**

The following example creates an envelope that ramps to 5V over a logarithmic curve:

```
TO.CV.SET 1 V 5
TO.CV.LOG 1 2
TO.ENV.ATT 1 500
TO.ENV.DEC.S 1 2
TO.ENV.ACT 1 1
```

When triggered (`TO.ENV.TRIG 1`), the envelope will rise to 5V over a half a second and then decay back to zero over two seconds. The curve used is 2, which covers 0V-5V.

If a curve is too small for the range being covered, values above the range will be limited to the range’s ceiling. In the above example, voltages above 5V will all return as 5V.

TO.CV.CALIB

- **TO.CV.CALIB x**

To calibrate your TXo outputs, follow these steps. Before you start, let your expander warm up for a few minutes. It won’t take long - but you want to make sure that it is calibrated at a more representative temperature.

Then, first adjust your offset (CV.OFF) until the output is at zero volts (0). For example:

```
CV.OFF 1 8
```

Once that output measures at zero volts, you want to lock it in as the calibration by calling the following operator:

```
CV.CALIB 1
```

You will find that the offset is now zero, but the output is at the value that you targeted during your prior adjustment. To reset to normal (and forget this calibration offset), use the `TO.CV.RESET` command.

TO.OSC

- **TO.OSC x y**

Targets oscillation for CV output x to y with the portamento rate determined by the `TO.OSC.SLEW` value. y is 1V/oct translated from the standard range (1-16384). A

value of 0 disables oscillation; CV amplitude is used as the peak for oscillation and needs to be > 0 for it to be perceivable.

Setting an OSC frequency greater than zero for a CV output will start that output oscillating. It will swing its voltage between to the current CV value and its polar opposite. For example:

```
TO.CV 1 V 5
TO.OSC 1 N 69
```

This will emit the audio-rate note A (at 440Hz) swinging from '+5V' to '-5V'. The CV value acts as an amplitude control. For example:

```
TO.CV.SLEW.M 1 1
TO.CV 1 V 10
```

This will cause the oscillations to gradually increase in amplitude from 5V to 10V over a period of one minute.

IMPORANT: if you do not set a CV value, the oscillator will not emit a signal.

If you want to go back to regular CV behavior, you need to set the oscillation frequency to zero. E.g. `TO.OSC 1 0`. You can also initialize the CV output with `TO.CV.INIT 1`, which resets all of its settings back to start-up default.

TO.OSC.SET

- **TO.OSC.SET x y**

Set oscillation for CV output x to y (ignores CV.OSC.SLEW.) y is 1V/oct translated from the standard range (1-16384); a value of 0 disables oscillation. CV amplitude is used as the peak for oscillation and needs to be > 0 for it to be perceivable.

TO.OSC.QT

- **TO.OSC.QT x y**

Targets oscillation for CV output x to y with the portamento rate determined by the TO.OSC.SLEW value. y is 1V/oct translated from the standard range (1-16384) and quantized to current OSC.SCALE. A value of 0 disables oscillation; CV amplitude is used as the peak for oscillation and needs to be > 0 for it to be perceivable.

TO.OSC.QT.SET

- **TO.OSC.QT.SET x y**

Set oscillation for CV output x to the 1V/oct value y (ignores CV.OSC.SLEW.) y is 1v/oct translated from the standard range (1-16384) and quantized to current OSC.SCALE. A value of 0 disables oscillation; CV amplitude is used as the peak for oscillation and needs to be > 0 for it to be perceivable.

TO.OSC.N

- **TO.OSC.N x y**

Targets oscillation for CV output x to note y with the portamento rate determined by the TO.OSC.SLEW value. See quantization scale reference for y; CV amplitude is used as the peak for oscillation and needs to be > 0 for it to be perceivable.

TO.OSC.N.SET

- **TO.OSC.N.SET x y**

Sets oscillation for CV output x to note y (ignores CV.OSC.SLEW.) See quantization scale reference for y; CV amplitude is used as the peak for oscillation and needs to be > 0 for it to be perceivable.

TO.OSC.FQ

- **TO.OSC.FQ x y**

Targets oscillation for CV output x to frequency y with the portamento rate determined by the TO.OSC.SLEW value. y is in Hz; a value of 0 disables oscillation. CV amplitude is used as the peak for oscillation and needs to be > 0 for it to be perceivable.

TO.OSC.FQ.SET

- **TO.OSC.FQ.SET x y**

Sets oscillation for CV output x to frequency y (ignores CV.OSC.SLEW.) y is in Hz; a value of 0 disables oscillation. CV amplitude is used as the peak for oscillation and needs to be > 0 for it to be perceivable.

TO.OSC.LFO

- **TO.OSC.LFO x y**

Targets oscillation for CV output x to LFO frequency y with the portamento rate determined by the TO.OSC.SLEW value. y is in mHz (millihertz: 10^-3 Hz); a value of 0 disables oscillation. CV amplitude is used as the peak for oscillation and needs to be > 0 for it to be perceivable.

TO.OSC.LFO.SET

- **TO.OSC.LFO.SET x y**

Sets oscillation for CV output x to LFO frequency y (ignores CV.OSC.SLEW.) y is in mHz (millihertz: 10^-3 Hz); a value of 0 disables oscillation. CV amplitude is used as the peak for oscillation and needs to be > 0 for it to be perceivable.

TO.OSC.SCALE

- **TO.OSC.SCALE x y**

Quantization Scales

0. Equal Temperament [DEFAULT]
1. 12-tone Pythagorean scale
2. Vallotti & Young scale (Vallotti version) also known as Tartini-Vallotti (1754)
3. Andreas Werckmeister's temperament III (the most famous one, 1681)
4. Wendy Carlos' Alpha scale with perfect fifth divided in nine
5. Wendy Carlos' Beta scale with perfect fifth divided by eleven
6. Wendy Carlos' Gamma scale with third divided by eleven or fifth by twenty
7. Carlos Harmonic & Ben Johnston's scale of 'Blues' from Suite f.micr.piano (1977) & David Beardsley's scale of 'Science Friction'
8. Carlos Super Just
9. Kurzweil "Empirical Arabic"
10. Kurzweil "Just with natural b7th", is Sauveur Just with 7/4
11. Kurzweil "Empirical Bali/Java Harmonic Pelog"
12. Kurzweil "Empirical Bali/Java Slendro, Siam 7"
13. Kurzweil "Empirical Tibetian Ceremonial"
14. Harry Partch's 43-tone pure scale
15. Partch's Indian Chromatic, Exposition of Monophony, 1933.
16. Partch Greek scales from "Two Studies on Ancient Greek Scales" on black/white

TO.OSC.WAVE

- **TO.OSC.WAVE x y**

There are 45 different waveforms, values translate to sine (0), triangle (100), saw (200), pulse (300) all the way to random/noise (4500). Oscillator shape between values is a blend of the pure waveforms.

TO.OSC.RECT

- **TO.OSC.RECT x y**

The rectification command performs a couple of levels of rectification based on how you have it set. The following values for y work as follows:

- y = 2: "full-positive" - inverts negative values, making them positive
- y = 1: "half-positive" - omits all negative values (values below zero are set to zero)
- y = 0: no rectification (default)
- y = -1: "half-negative" - omits all positive values (values above zero are set to zero)
- y = -2: "full-negative" - inverts positive values, making them negative

TO.OSC.SLEW

- **TO.OSC.SLEW x y**

This parameter acts as a frequency slew for the targeted CV output. It allows you to gradually slide from one frequency to another, creating a portamento like effect. It is also great for smoothing transitions between different LFO rates on the oscillator. For example:

```
TO.CV 1 V 5
TO.OSC.SLEW 1 30000
TO.OSC.LFO.SET 1 1000
TO.OSC.LFO 1 100
```

This will start an LFO on CV 1 with a rate of 1000mHz. Then, over the next 30 seconds, it will gradually decrease in rate to 100mHz.

TO.OSC.CTR

- **TO.OSC.CTR x y**

For example, to create a sine wave that is centered at 2.5V and swings up to +5V and down to 0V, you would do this:

```
TO.CV 1 VV 250
TO.OSC.CTR 1 VV 250
TO.OSC.LFO 1 500
```

TO.ENV.ACT

- **TO.ENV.ACT x y**

This setting activates (1) or deactivates (0) the envelope generator on CV output y. The envelope generator is dependent on the current voltage setting for the output. Upon activation, the targeted output will go to zero. Then, when triggered (TO.ENV.TRIG), it will ramp the voltage from zero to the currently set peak voltage (TO.CV) over the attack time (TO.ENV.ATT) and then decay back to zero over the decay time (TO.ENV.DEC). For example:

```
TO.CV.SET 1 V 8
TO.ENV.ACT 1 1
TO.ENV.ATT.S 1 1
TO.ENV.DEC.S 1 30
```

This will initialize the CV 1 output to have an envelope that will ramp to +8V over one second and decay back to zero over thirty seconds. To trigger the envelope, you need to send the trigger command TO.ENV.TRIG 1. Envelopes currently re-trigger from the start of the cycle.

To return your CV output to normal function, either deactivate the envelope (TO.ENV.ACT 1 0) or reinitialize the output (TO.CV.INIT 1).

TO.ENV

- **TO.ENV x y**

This parameter essentially allows output x to act as a gate between the 0 and 1 state. Changing this value from 0 to 1 causes the envelope to trigger the attack phase and hold at the peak CV value; changing this value from 1 to 0 causes the decay stage of the envelope to be triggered.

TO.ENV.EOR

- **TO.ENV.EOR x n**

Fires a PULSE at the End of Rise to the unit-local trigger output n for the envelope on CV output x; n refers to trigger output 1-4 on the same TXo as CV output x.

The most important thing to know with this operator is that you can only cause the EOR trigger to fire on the same device as the TXo with the envelope. For this command, the outputs are numbered LOCALLY to the unit with the envelope.

For example, if you have an envelope running on your second TXo, you can only send the EOR pulse to the four outputs on that device:

```
TO.ENV.EOR 5 1
```

This will cause the first output on TXo #2 (`TO.TR 5`) to pulse after the envelope's attack segment.

TO.ENV.EOC

- **TO.ENV.EOC x n**

Fires a PULSE at the End of Cycle to the unit-local trigger output n for the envelope on CV output x. n refers to trigger output 1-4 on the same TXo as CV output 'y'.

The most important thing to know with this operator is that you can only cause the EOC trigger to fire on the same device as the TXo with the envelope. For this command, the outputs are numbered LOCALLY to the unit with the envelope.

For example, if you have an envelope running on your second TXo, you can only send the EOC pulse to the four outputs on that device:

```
TO.ENV.EOC 5 1
```

This will cause the first output on TXo #2 (`TO.TR 5`) to pulse after the envelope's decay segment.

TO.ENV.LOOP

- **TO.ENV.LOOP x y**

Causes the envelope on CV output x to loop for y times. A y of 0 will cause the envelope to loop infinitely; setting y to 1 (default) disables looping and (if currently looping) will cause it to finish its current cycle and cease.

Crow

OP	OP *(set)*	*(aliases)*	Description
CROW.SEL x			Sets target crow unit (1 (default), to 4).
CROWN: ...			Send following CROW OPs to all units starting with selected unit.
CROW1: ...			Send following CROW OPs to unit 1 ignoring the currently selected unit.
CROW2: ...			Send following CROW OPs to unit 2 ignoring the currently selected unit.
CROW3: ...			Send following CROW OPs to unit 3 ignoring the currently selected unit.
CROW4: ...			Send following CROW OPs to unit 4 ignoring the currently selected unit.
CROW.V x y			Sets output x to value y. Use V y for volts.
CROW.SLEW x y			Sets output x slew rate to y milliseconds.
CROW.C1 x			Calls the function `ii.self.call1(x)` on crow.
CROW.C2 x y			Calls the function `ii.self.call2(x, y)` on crow.
CROW.C3 x y z			Calls the function `ii.self.call3(x, y, z)` on crow.
CROW.C4 x y z t			Calls the function `ii.self.call4(x, y, z, t)` on crow.
CROW.RST			Calls the function `crow.reset()` returning crow to default state.
CROW.PULSE x y z t			Creates a trigger pulse on output x with duration y (ms) to voltage z with polarity `t`.
CROW.AR x y z t			Creates an envelope on output x, rising in y ms, falling in z ms, and reaching height `t`.

OP	OP *(set)*	*(aliases)*	Description
`CROW.LFO x y z t`			Starts an envelope on output `x` at rate `y` where `0 = 1Hz` with 1v/octave scaling. `z` sets amplitude and `t` sets skew for assymetrical triangle waves.
`CROW.IN x`			Gets voltage at input `x`.
`CROW.OUT x`			Gets voltage of output `x`.
`CROW.Q0`			Returns the result of calling the function `crow.self.query0()`.
`CROW.Q1 x`			Returns the result of calling the function `crow.self.query1(x)`.
`CROW.Q2 x y`			Returns the result of calling the function `crow.self.query2(x, y)`.
`CROW.Q3 x y z`			Returns the result of calling the function `crow.self.query3(x, y, z)`.

W/

More extensively covered in the W/ Documentation[31].

OP	OP *(set)*	*(aliases)*	Description
`WS.PLAY x`			Set playback state and direction. `0` stops playback. 1 sets forward motion, while `-1` plays in reverse
`WS.REC x`			Set recording mode. `0` is playback only. 1 sets overdub mode for additive recording. `-1` sets overwrite mode to replace the tape with your input
`WS.CUE x`			Go to a cuepoint relative to the playhead position. `0` retriggers the current location. 1 jumps to the next cue forward. `-1` jumps to the previous cue in the reverse. These actions are relative to playback direction such that `0` always retriggers the most recently passed location
`WS.LOOP x`			Set the loop state on/off. `0` is off. Any other value turns loop on

[31] https://www.whimsicalraps.com/pages/w-type

W/2.0

More extensively covered in the W/ Documentation[32].

There are separate ops for each supported algorithm: delay, synth, tape. Two units can be connected using a different i2c address (refer to the official documentation for more details). The following section describes ops that control which unit is selected. These ops apply to all algorithms.

OP	OP *(set)*	*(aliases)*	Description
`W/.SEL x`			Sets target W/2.0 unit (1 = primary, 2 = secondary).
`W/1: ...`			Send following W/2.0 OPs to unit 1 ignoring the currently selected unit.
`W/2: ...`			Send following W/2.0 OPs to unit 2 ignoring the currently selected unit.

[32]https://www.whimsicalraps.com/pages/w-type

W/2.0 tape

Tape mode of W/ eurorack module.

More extensively covered in the W/ Documentation[33].

OP	OP *(set)*	*(aliases)*	Description
`W/T.REC active`			Sets recording state to `active` (s8)
`W/T.PLAY playback`			Set the playback state. -1 will flip playback direction (s8)
`W/T.REV`			Reverse the direction of playback
`W/T.SPEED speed deno`			Set speed as a rate, or ratio. Negative values are reverse (s16V)
`W/T.FREQ freq`			Set speed as a `frequency` (s16V) style value. Maintains reverse state
`W/T.ERASE.LVL level`			Strength of erase head when recording. 0 is overdub, 1 is overwrite. Opposite of feedback (s16V)
`W/T.MONITOR.LVL gain`			Level of input passed directly to output (s16V)
`W/T.REC.LVL gain`			Level of input material recorded to tape (s16V)
`W/T.ECHOMODE is_echo`			Set to 1 to playback before erase. 0 (default) erases first (s8)
`W/T.LOOP.START`			Set the current time as the beginning of a loop
`W/T.LOOP.END`			Set the current time as the loop end, and jump to start
`W/T.LOOP.ACTIVE state`			Set the state of looping (s8)
`W/T.LOOP.SCALE scale`			Mul(Positive) or Div(Negative) loop brace by arg. Zero resets to original window (s8)
`W/T.LOOP.NEXT direction`			Move loop brace forward/backward by length of loop. Zero jumps to loop start (s8)
`W/T.TIME seconds sub`			Move playhead to an arbitrary location on tape (s16)

[33] https://www.whimsicalraps.com/pages/w-type

OP	OP *(set)*	*(aliases)*	Description
W/T.SEEK seconds sub			Move playhead relative to current position (s16)
W/T.CLEARTAPE			WARNING! Erases all recorded audio on the tape!

W/2.0 delay

Delay mode of W/ eurorack module.

More extensively covered in the W/ Documentation[34].

OP	OP *(set)*	*(aliases)*	Description
`W/D.FBK level`			amount of feedback from read head to write head (s1
`W/D.MIX fade`			fade from dry to delayed signal
`W/D.FILT cutoff`			centre frequency of filter in feedback loop (s16V)
`W/D.FREEZE is_active`			deactivate record head to freeze the current buffer (s8
`W/D.TIME seconds`			set delay buffer length in `seconds` (s16V), when rate
`W/D.LEN count divisions`			set buffer loop length as a fraction of buffer time (u8)
`W/D.POS count divisions`			set loop start location as a fraction of buffer time (u8
`W/D.CUT count divisions`			jump to loop location as a fraction of loop length (u8)
`W/D.FREQ.RNG freq_range`			TBD (s8)
`W/D.RATE multiplier`			direct `multiplier` (s16V) of tape speed
`W/D.FREQ volts`			manipulate tape speed with musical values (s16V)
`W/D.CLK`			receive clock pulse for synchronization
`W/D.CLK.RATIO mul div`			set clock pulses per buffer time, with clock `mul/div`
`W/D.PLUCK volume`			pluck the delay line with noise at volume (s16V)
`W/D.MOD.RATE rate`			set the multiplier for the modulation rate (s16V)
`W/D.MOD.AMT amount`			set the `amount` (s16V) of delay line modulation to be

[34] https://www.whimsicalraps.com/pages/w-type

W/2.0 synth

Synth mode of W/ eurorack module.

More extensively covered in the W/ Documentation[35].

OP	OP *(set)*	*(aliases)*	Description
W/S.PITCH voice pitch			set `voice` (s8) to `pitch` (s16V) in volts-per-octave
W/S.VEL voice velocity			strike the vactrol of `voice` (s8) at `velocity` (s16V) in volts
W/S.VOX voice pitch velocity			set `voice` (s8) to `pitch` (s16V) and strike the vactrol at `velocity` (s16V)
W/S.NOTE pitch level			dynamically assign a voice, set to `pitch` (s16V), strike with `velocity`(s16V)
W/S.POLY pitch level			As `W/S.NOTE` but across dual W/. Switches between primary and secondary units every 4 notes or until reset using `W/S.POLY.RESET`.
W/S.POLY.RESET			Resets `W/S.POLY` note count.
W/S.AR.MODE is_ar			in attack-release mode, all notes are `plucked` and no `release` is required'
W/S.LPG.TIME time			vactrol `time` (s16V) constant. -5=drones, 0=vtl5c3, 5=blits
W/S.LPG.SYM symmetry			vactrol attack-release ratio. -5=fastest attack, 5=long swells (s16V)
W/S.CURVE curve			cross-fade waveforms: -5=square, 0=triangle, 5=sine (s16V)
W/S.RAMP ramp			waveform symmetry: -5=rampwave, 0=triangle, 5=sawtooth (NB: affects FM tone)
W/S.FM.INDEX index			amount of FM modulation. -5=negative, 0=minimum, 5=maximum (s16V)
W/S.FM.RATIO num den			ratio of the FM modulator to carrier as a ratio. floating point values up to 20.0 supported (s16V)

[35]https://www.whimsicalraps.com/pages/w-type

OP	OP *(set)*	*(aliases)*	Description
`W/S.FM.ENV amount`			amount of vactrol envelope applied to fm index, -5 to +5 (s16V)
`W/S.PATCH jack param`			patch a hardware `jack` (s8) to a `param` (s8) destination
`W/S.VOICES count`			set number of polyphonic voices to allocate. use 0 for unison mode (s8)

Disting EX

The Expert Sleepers Disting EX is a multifunction Eurorack module. It can communicate with the Teletype allowing you to select algorithms, save and load presets, control parameters, play notes and send MIDI and Select Bus messages. Up to four devices can be connected. The Disting I2C address must be set to 65 (66..68 for additional modules).

OP	OP *(set)*	*(aliases)*	Description
EX	**EX x**		get or set currently selected unit to x (1-4)
EX1: ...			send following Disting ops to unit 1 ignoring the currently selected unit
EX2: ...			send following Disting ops to unit 2 ignoring the currently selected unit
EX3: ...			send following Disting ops to unit 3 ignoring the currently selected unit
EX4: ...			send following Disting ops to unit 4 ignoring the currently selected unit
EX.PRESET	**EX.PRESET x**	**EX.PRE**	load preset x or get the currently loaded preset
EX.SAVE x			save to preset x
EX.RESET			reset the currently loaded preset
EX.ALG	**EX.ALG x**	**EX.A**	get or set the current algorithm to x (single algorithms only)
EX.CTRL x y		**EX.C**	set I2C controller x to value y
EX.PARAM x	**EX.PARAM x y**	**EX.P**	set parameter x to value y or get the current parameter value
EX.PV x y			set parameter x using a value determined by scaling y from 0..16384 range.
EX.MIN x			get the minimum possible value for parameter x
EX.MAX x			get the maximum possible value for parameter x
EX.VOX x y z		**EX.V**	send a note to voice x using pitch y and velocity z
EX.VOX.P x y		**EX.VP**	set voice x to pitch y
EX.VOX.O x		**EX.VO**	send a note off to voice x

OP	OP *(set)*	*(aliases)*	Description
EX.CH x		**EX.#**	select default note channel (for multi channel algorithms like Poly FM)
EX.NOTE x y		**EX.N**	send a note using pitch x and velocity y (voice allocated by the Disting)
EX.N# x y z			send a note to channel x using pitch y and velocity z (voice allocated by the Disting)
EX.NOTE.O x		**EX.NO**	send a note off using pitch x
EX.NO# x y			send a note off to channel x using pitch y
EX.ALLOFF		**EX.AO**	all notes off
EX.T x			send a trigger to voice x with medium velocity (use with SD Triggers algo)
EX.TV x y			send a trigger to voice x using velocity y (use with SD Triggers algo)
EX.REC x			control WAV recorder recording: 1 to start, 0 to stop
EX.PLAY x			control WAV recorder playback: 1 to start, 0 to stop
EX.AL.P x			set Augustus Loop pitch to value x
EX.AL.CLK			send clock to Augustus Loop
EX.LP x			get current state for loop x
EX.LP.REC x			toggle recording for loop x
EX.LP.PLAY x			toggle playback for loop x
EX.LP.CLR x			clear loop x
EX.LP.REV x			toggle reverse for loop x
EX.LP.REV? x			returns 1 if loop x is reversed, 0 otherwise
EX.LP.DOWN x			toggle octave down for loop x
EX.LP.DOWN? x			return 1 if loop x is transposed octave down, 0 otherwise
EX.M.CH	**EX.M.CH x**		get or set the currently selected MIDI channel (1-16)
EX.M.N x y			send MIDI Note On message for note x (0..127) and velocity y (0..127)
EX.M.N# x y z			send MIDI Note On message on channel x for note y (0..127) and velocity z (0..127)

OP	OP *(set)*	*(aliases)*	Description
EX.M.NO x			send MIDI Note off message for note x (0..127)
EX.M.NO# x y			send MIDI Note off message on channel x for note y (0..127)
EX.M.CC x y			send MIDI CC message for controller x (0..127) and value y (0..127)
EX.M.CC# x y z			send MIDI CC message on channel x for controller y (0..127) and value z (0..127)
EX.M.PB x			send MIDI Pitchbend message
EX.M.PRG x			send MIDI Program Change message
EX.M.CLK			send MIDI clock message
EX.M.START			send MIDI Start message
EX.M.STOP			send MIDI Stop message
EX.M.CONT			send MIDI Continue message
EX.SB.CH	**EX.SB.CH x**		get or set the currently selected Select Bus channel (1-16)
EX.SB.N x y			send Select Bus Note On message for note x (0..127) and velocity y (0..127)
EX.SB.NO x			send Select Bus Note off message for note x (0..127)
EX.SB.PB x			send Select Bus Pitchbend message
EX.SB.CC x y			send Select Bus CC message for controller x (0..127) and value y (0..127)
EX.SB.PRG x			send Select Bus Program Change message
EX.SB.CLK			send Select Bus clock message
EX.SB.START			send Select Bus Start message
EX.SB.STOP			send Select Bus Stop message
EX.SB.CONT			send Select Bus Continue message
EX.A1	**EX.A1 x**		get or set the left dual algorithm
EX.A2	**EX.A2 x**		get or set the right dual algorithm
EX.A12 x y			set both dual algorithms
EX.P1 x	**EX.P1 x y**		get left algorithm parameter x or set it to value y

OP	OP *(set)*	*(aliases)*	Description
EX.P2 x	**EX.P2 x y**		get right algorithm parameter x or set it to value y
EX.PV1 x y			set left algorithm parameter x using a value determined by scaling y from 0..16384 range
EX.PV2 x y			set right algorithm parameter x using a value determined by scaling y from 0..16384 range
EX.MIN1 x			get left algorithm parameter minimum value
EX.MAX1 x			get left algorithm parameter maximum value
EX.MIN2 x			get right algorithm parameter minimum value
EX.MAX2 x			get right algorithm parameter maximum value
EX.Z1	**EX.Z1 x**		get left Z knob value or set left Z parameter (0..127 range)
EX.ZO1			restore control for left Z knob and input
EX.Z2	**EX.Z2 x**		get right Z knob value or set right Z parameter (0..127 range)
EX.ZO2			restore control for right Z knob and input
EX.PRE1 x			load left preset from x slot
EX.PRE2 x			load right preset from x slot
EX.SAVE1 x			save left preset to x slot
EX.SAVE2 x			save right preset to x slot

EX

- **EX** / **EX x**

All Disting EX ops address the currently selected unit (unit 1 is selected by default), unless placed after EX1..EX4 modifiers. Use this op to select a unit or get the currently selected unit.

EX.CTRL

- **EX.CTRL x y**
- *alias*: **EX.C**

I2C controllers must be mapped to parameters on the Disting. Handy for controlling multiple parameters at once.

EX.PARAM

- **EX.PARAM x** / **EX.PARAM x y**
- *alias*: **EX.P**

This op sets the specified parameter to the exact value specified. If you want to use something like IN or PARAM to control a value, use EX.PV op instead as it will scale it properly for you.

EX.PV

- **EX.PV x y**

This op sets the specified parameter by scaling the provided value from 0..16384 range to the range used by the parameter. This op is useful when used in conjunction with something like PARAM: EX.PV x PARAM will allow you to use the param knob to control full range of parameter x.

EX.VOX

- **EX.VOX x y z**
- *alias*: **EX.V**

This op will trigger notes when using SD Multisample or SD Triggers algorithms. Please note chord/arpeggio Disting functionality is only available when using voices allocated by the Disting itself (see EX.NOTE op).

EX.AL.P

- **EX.AL.P x**

Parameter 32 (Pitch CV Input) must be set to None in order for this to work. Pitch value uses the same scale as other pitch related ops, so you can use N and related ops to convert semitones to the actual pitch value.

EX.LP

- **EX.LP x**

0 - initial state 1 - recording 2 - recording extra material for crossfade 3 - playback 4 - overdub 5 - paused / muted 6 - fading out towards pause

EX.M.CH

- **EX.M.CH** / **EX.M.CH x**

All MIDI ops will use the currently selected channel except ops that use a channel parameter. MIDI ops require a MIDI breakout.

EX.SB.CH

- **EX.SB.CH** / **EX.SB.CH x**

All Select Bus ops that use a channel will use the currently selected channel. Select Bus settings must be properly configured on the Disting.

EX.Z1

- **EX.Z1** / **EX.Z1 x**

Setting Z parameter value will disengage Z knob and input, use EX.Z01 to restore control.

EX.Z2

- **EX.Z2** / **EX.Z2 x**

Setting Z parameter value will disengage Z knob and input, use EX.Z02 to restore control.

Matrixarchate

The SSSR Labs SM010 Matrixarchate is a 16x8 IO Sequenceable Matrix Signal Router.

OP	OP *(set)*	*(aliases)*	Description
MA.SELECT x			select the default matrixarchate module, default 1
MA.STEP			advance program sequencer
MA.RESET			reset program sequencer
MA.PGM pgm			select the current program (1-based)
MA.ON x y			connect row x and column y in the current program (rows/columns are 0-based)
MA.PON pgm x y			connect row x and column y in program pgm
MA.OFF x y			disconnect row x and column y in the current program
MA.POFF x y pgm			connect row x and column y in program pgm
MA.SET x y state			set the connection at row x and column y to state (1 - on, 0 - off)
MA.PSET pgm x y state			set the connection at row x and column y in program pgm to state (1 - on, 0 - off)
MA.COL col	**MA.COL col value**		get or set column col (as a 16 bit unsigned value where each bit represents a connection)
MA.PCOL pgm col	**MA.PCOL pgm col value**		get or set column col in program pgm
MA.ROW row	**MA.ROW row value**		get or set row row
MA.PROW pgm row	**MA.PROW pgm row value**		get or set row row in program pgm
MA.CLR			clear all connections
MA.PCLR pgm			clear all connections in program pgm

i2c2midi

i2c2midi is a DIY open source 2 HP Teletype Expander that speaks I2C and MIDI. It bridges the gap between monome Teletype and external MIDI-enabled devices, using I2C: It receives I2C messages from Teletype and converts them to MIDI notes, MIDI CC messages and other MIDI messages to control external devices like synths and effects; it receives MIDI messages from external MIDI controllers and stores the values internally, which can be requested at any time by Teletype via I2C. For more information: https://github.com/attowatt/i2c2midi

OP	OP *(set)*	*(aliases)*	Description
I2M.CH	**I2M.CH x**	**I2M.#**	Get currently set MIDI channel / Set MIDI channel x (1..16 for TRS, 17..32 for USB) for MIDI out
I2M.TIME	**I2M.TIME x**	**I2M.T**	Get current note duration / Set note duration of MIDI notes to x ms (0..32767) for current channel
I2M.T# ch	**I2M.T# ch x**		Get current note duration / Set note duration of MIDI notes to x ms (0..32767) for channel ch (0..32).
I2M.SHIFT	**I2M.SHIFT x**	**I2M.S**	Get current transposition / Set transposition of MIDI notes to x semitones (-127..127) for current channel
I2M.S# ch	**I2M.S# ch x**		Get current transposition / Set transposition of MIDI notes to x semitones (-127..127) for channel ch (0..32)
I2M.MIN x y			Set minimum note number for MIDI notes to x (0..127), using mode y (0..3), for current channel
I2M.MIN# ch x y			Set minimum note number for MIDI notes to x (0..127), using mode y (0..3), for channel ch (0..32)
I2M.MAX x y			Set maximum note number for MIDI notes to x (0..127), using mode y (0..3), for current channel
I2M.MAX# ch x y			Set maximum note number for MIDI notes to x (0..127), using mode y (0..3), for channel ch (0..32)

OP	OP *(set)*	*(aliases)*	Description
I2M.REP	**I2M.REP x**		Get current repetition / Set repetition of MIDI notes to x repetitions (1..127) for current channel
I2M.REP# ch x			Get current repetition / Set repetition of MIDI notes to x repetitions (1..127) for channel ch (0..32)
I2M.RAT	**I2M.RAT x**		Get current ratcheting / Set ratcheting of MIDI notes to x ratchets (1..127) for current channel
I2M.RAT# ch x			Get current ratcheting / Set ratcheting of MIDI notes to x ratchets (1..127) for channel ch (0..32)
I2M.MUTE	**I2M.MUTE x**		Get mute state / Set mute state of current MIDI channel to x (0..1)
I2M.MUTE#	**I2M.MUTE# ch x**		Get mute state / Set mute state of MIDI channel ch to x (0..1)
I2M.SOLO	**I2M.SOLO x**		Get solo state / Set solo state of current MIDI channel to x (0..1)
I2M.SOLO#	**I2M.SOLO# ch x**		Get solo state / Set solo state of MIDI channel ch to x (0..1)
I2M.NOTE x y		**I2M.N**	Send MIDI Note On message for note number x (0..127) with velocity y (1..127) on current channel
I2M.N# ch x y			Send MIDI Note On message for note number x (0..127) with velocity y (1..127) on channel ch (1..32)
I2M.NOTE.O x		**I2M.NO**	Send a manual MIDI Note Off message for note number x (0..127)
I2M.NO# ch x			Send a manual MIDI Note Off message for note number x (0..127) on channel ch (1..32)
I2M.NT x y z			Send MIDI Note On message for note number x (0..127) with velocity y (1..127) and note duration z ms (0..32767)

OP	OP *(set)*	*(aliases)*	Description
I2M.NT# ch x y z			Send MIDI Note On message for note number x (0..127) with velocity y (1..127) and note duration z ms (0..32767) on channel ch (1..32)
I2M.CC x y			Send MIDI CC message for controller x (0..127) with value y (0..127)
I2M.CC# ch x y			Send MIDI CC message for controller x (0..127) with value y (0..127) on channel ch (1..32)
I2M.CC.SET x y			Send MIDI CC message for controller x (0..127) with value y (0..127), bypassing any slew settings
I2M.CC.SET# ch x y			Send MIDI CC message for controller x (0..127) with value y (0..127) on channel ch (1..32), bypassing any slew settings
I2M.CCV x y			Send MIDI CC message for controller x (0..127) with volt value y (0..16383, 0..+10V)
I2M.CCV# ch x y			Send MIDI CC message for controller x (0..127) with volt value y (0..16383, 0..+10V) on channel ch (1..32)
I2M.CC.OFF x	**I2M.CC.OFF x y**		Get current offset / Set offset of values of controller x (0..127) to y (-127..127)
I2M.CC.OFF# ch x	**I2M.CC.OFF# ch x y**		Get current offset / Set offset of values of controller x (0..127) to y (-127..127) for channel ch (1..32)
I2M.CC.SLEW x	**I2M.CC.SLEW x y**		Get current slew time for controller x / Set slew time for controller x (0..127) to y ms (0..32767)
I2M.CC.SLEW# ch x	**I2M.CC.SLEW# ch x y**		Get current slew time for controller x / Set slew time for controller x (0..127) to y ms (0..32767) for channel ch (1..32)
I2M.NRPN x y z			Send MIDI NRPN message (high-res CC) for parameter MSB x and LSB y with value y (0..16383)

OP	OP *(set)*	*(aliases)*	Description
I2M.NRPN# ch x y z			Send MIDI NRPN message (high-res CC) for parameter MSB x and LSB y with value y (0..16383) on channel ch (1..32)
I2M.NRPN.OFF x y	**I2M.NRPN.OFF x y z**		Get current offset / Set offset of values of NRPN messages to z (-16384..16383)
I2M.NRPN.OFF# ch x y	**I2M.NRPN.OFF# ch x y z**		Get current offset / Set offset of values of NRPN messages to z (-16384..16383) for channel ch (1..32)
I2M.NRPN.SLEW x y	**I2M.NRPN.SLEW x y z**		Get current slew time / Set slew time for NRPN messages to z ms (0..32767)
I2M.NRPN.SLEW# ch x y	**I2M.NRPN.SLEW# ch x y z**		Get current slew time / Set slew time for NRPN messages to z ms (0..32767) for channel ch (1..32)
I2M.NRPN.SET x y z			Send MIDI NRPN message for parameter MSB x and LSB y with value y (0..16383), bypassing any slew settings
I2M.NRPN.SET# ch x y z			Send MIDI NRPN message for parameter MSB x and LSB y with value y (0..16383) on channel ch (1..32), bypassing any slew settings
I2M.PRG x			Send MIDI Program Change message for program x (0..127)
I2M.PB x			Send MIDI Pitch Bend message with value x (-8192..8191)
I2M.AT x			Send MIDI After Touch message with value x (0..127)
I2M.CLK			Send MIDI Clock message, this still needs improvement …
I2M.START			Send MIDI Clock Start message
I2M.STOP			Send MIDI Clock Stop message
I2M.CONT			Send MIDI Clock Continue message
I2M.CHORD x y z		**I2M.C**	Play chord x (1..8) with root note y (-127..127) and velocity z (1..127)
I2M.C# ch x y z			Play chord x (1..8) with root note y (-127..127) and velocity z (1..127) on channel ch (1..32)

OP	OP *(set)*	*(aliases)*	Description
I2M.C.ADD x y		**I2M.C+**	Add relative note y (-127..127) to chord x (0..8), use x = 0 to add to all chords
I2M.C.RM x y		**I2M.C-**	Remove note y (-127..127) from chord x (0..8), use x = 0 to remove from all chords
I2M.C.INS x y z			Add note z (-127..127) to chord x (0..8) at index y (0..7), with z relative to the root note; use x = 0 to insert into all chords
I2M.C.DEL x y			Delete note at index y (0..7) from chord x (0..8), use x = 0 to delete from all chords
I2M.C.SET x y z			Set note at index y (0..7) in chord x (0..8) to note z (-127..127), use x = 0 to set in all chords
I2M.C.B x y			Clear and define chord x (0..8) using reverse binary notation (R...)
I2M.C.CLR x			Clear chord x (0..8), use x = 0 to clear all chords
I2M.C.L x	**I2M.C.L x y**		Get current length / Set length of chord x (0..8) to y (1..8), use x = 0 to set length of all chords
I2M.C.SC x y			Set scale for chord x (0..8) based on chord y (0..8), use x = 0 to set for all chords, use y = 0 to remove scale
I2M.C.REV x y			Set reversal of notes in chord x (0..8) to y. y = 0 or an even number means not reversed, y = 1 or an uneven number means reversed. Use x = 0 to set for all chords.
I2M.C.ROT x y			Set rotation of notes in chord x (0..8) to y steps (-127..127), use x = 0 to set for all chords
I2M.C.TRP x y			Set transposition of chord x (0..8) to y (-127..127), use x = 0 to set for all chords
I2M.C.DIS x y z			Set distortion of chord x (0..8) to y (-127..127) with anchor point z (0..16), use x = 0 to set for all chords

OP	OP *(set)*	*(aliases)*	Description
I2M.C.REF x y z			Set reflection of chord x (0..8) to y iterations (-127..127) with anchor point z (0..16), use x = 0 to set for all chords
I2M.C.INV x y			Set inversion of chord x (0..8) to y (-32..32), use x = 0 to set for all chords
I2M.C.STR x y			Set strumming of chord x (0..8) to x ms (0..32767), use x = 0 to set for all chords
I2M.C.VCUR w x y z		**I2M.C.V~**	Set velocity curve for chord w (0..8) with curve type x (0..5), start value y% (0..32767) and end value z% (0..32767), use w = 0 to set for all chords, use x = 0 to turn off
I2M.C.TCUR w x y z		**I2M.C.T~**	Set time curve to strumming for chord w (0..8) with curve type x (0..5), start value y% (0..32767) and end value z% (0..32767), use w = 0 to set for all chords, use x = 0 to turn off
I2M.C.DIR x y			Set play direction for chord x (0..8) to direction y (0..8)
I2M.C.QN x y z			Get the transformed note number of a chord note for chord x (1..8) with root note y (-127..127) at index z (0..7)
I2M.C.QV x y z			Get the transformed note velocity of a chord note for chord x (1..8) with root velocity y (1..127) at index z (0..7)
I2M.B.R x			Turn recording of notes into the buffer on or off
I2M.B.L x			Set the length of the buffer to x ms (0..32767)
I2M.B.START x			Add an offset of x ms (0..32767) to the start of the buffer
I2M.B.END x			Add a negative offset of x ms (0..32767) to the end of the buffer
I2M.B.DIR x			Set the play direction x (0..2) of the buffer

OP	OP *(set)*	*(aliases)*	Description
I2M.B.SPE x			Set the playing speed x (1..32767) of the buffer. x = 100 is equivalent to 'normal speed', x = 50 means double the speed, x = 200 means half the speed, etc.
I2M.B.FB x			Set the feedback length x (0..255) of the buffer
I2M.B.NSHIFT x			Set the note shift of recorded notes to x semitones (-127..127)
I2M.B.VSHIFT x			Set the velocity shift of recorded notes to x (-127..127)
I2M.B.TSHIFT x			Set the note duration shift ('time shift') of recorded notes to x ms (-16384..16383)
I2M.B.NOFF x			Set the note offset of recorded notes to x semitones (-127..127)
I2M.B.VOFF x			Set the velocity offset of recorded notes to x (-127..127)
I2M.B.TOFF x			Set the note duration offset ('time offset') of recorded notes to x ms (-16384..16383)
I2M.B.CLR			Clear the buffer, erasing all recorded notes in the buffer
I2M.B.MODE x			Set the buffer mode to x (0..1). 1) Digital 2) Tape
I2M.Q.CH	**I2M.Q.CH x**	**I2M.Q.#**	Get currently set MIDI channel / Set MIDI channel x (1..16) for MIDI in
I2M.Q.LATCH x			Turn on or off 'latching' for MIDI notes received via MIDI in
I2M.Q.NOTE x		**I2M.Q.N**	Get x (0..7) last note number (0..127) received via MIDI in
I2M.Q.VEL x		**I2M.Q.V**	Get x (0..7) last note velocity (1..127) received via MIDI in
I2M.Q.CC x			Get current value (0..127) of controller x (0..127) received via MIDI in
I2M.Q.LCH			Get the latest channel (1..16) received via MIDI in
I2M.Q.LN			Get the note number (0..127) of the latest Note On received via MIDI in

OP	OP *(set)*	*(aliases)*	Description
I2M.Q.LV			Get the velocity (1..127) of the latest Note On received via MIDI in
I2M.Q.LO			Get the note number (0..127) of the latest Note Off received via MIDI in
I2M.Q.LC			Get the latest controller number (0..127) received via MIDI in
I2M.Q.LCC			Get the latest controller value (0..127) received via MIDI in
I2M.PANIC			Send MIDI Note Off messages for all notes on all channels, and reset note duration, shift, repetition, ratcheting, min/max

I2M.CH

- **I2M.CH** / **I2M.CH x**
- *alias*: **I2M.#**

Get currently set MIDI channel / Set MIDI channel x (1..16 for TRS, 17..32 for USB) for MIDI out. Use MIDI channels 1-16 for TRS output, 17-32 for USB output. Default is `x = 1`.

I2M.TIME

- **I2M.TIME** / **I2M.TIME x**
- *alias*: **I2M.T**

Get current note duration / Set note duration of MIDI notes to x ms (0..32767) for current channel. Based on note duration, i2c2midi will send a MIDI Note Off message automatically. Set `x = 0` to deactivate automatic Note Off messages. Default is `x = 100`.

I2M.T#

- **I2M.T# ch** / **I2M.T# ch x**

Get current note duration / Set note duration of MIDI notes to x ms (0..32767) for channel ch (0..32). Use `ch = 0` to set for all channels.

I2M.SHIFT

- **I2M.SHIFT** / **I2M.SHIFT x**
- *alias*: **I2M.S**

Get current transposition / Set transposition of MIDI notes to x semitones (-127..127) for current channel. Default is `x = 0`.

I2M.S#

- **I2M.S# ch** / **I2M.S# ch x**

Get current transposition / Set transposition of MIDI notes to x semitones (-127..127) for channel `ch` (0..32). Use `ch = 0` to set for all channels."

I2M.MIN

- **I2M.MIN x y**

Set minimum note number for MIDI notes to x (0..127), using mode y (0..3), for current channel. Default is `x = 0` and `y = 0`. The following modes are available for notes lower than the minimum: 0) Ignore notes 1) Clamp notes 2) Fold back notes by one octave 3) Fold back notes by multiple octaves.

I2M.MIN#

- **I2M.MIN# ch x y**

Set minimum note number for MIDI notes to x (0..127), using mode y (0..3), for channel `ch` (0..32). Use `ch = 0` to set for all channels.

I2M.MAX

- **I2M.MAX x y**

Set maximum note number for MIDI notes to x (0..127), using mode y (0..3), for current channel. Default is `x = 0` and `y = 0`. The following modes are available for notes higher than the maximum: 0) Ignore notes 1) Clamp notes 2) Fold back notes by one octave 3) Fold back notes by multiple octaves.

I2M.MAX#

- **I2M.MAX# ch x y**

Set maximum note number for MIDI notes to x (0..127), using mode y (0..3), for channel `ch` (0..32). Use `ch = 0` to set for all channels.

I2M.REP

- **I2M.REP** / **I2M.REP x**

Get current repetition / Set repetition of MIDI notes to x repetitions (1..127) for current channel. Set `x = 1` for no repetitions. Default is `x = 1`.

I2M.REP#

- **I2M.REP# ch x**

Get current repetition / Set repetition of MIDI notes to x repetitions (1..127) for channel ch (0..32). Use `ch = 0` to set for all channels.

I2M.RAT

- **I2M.RAT** / **I2M.RAT x**

Get current ratcheting / Set ratcheting of MIDI notes to x ratchets (1..127) for current channel. Set `x = 1` for no ratcheting. Default is `x = 1`.

I2M.RAT#

- **I2M.RAT# ch x**

Get current ratcheting / Set ratcheting of MIDI notes to x ratchets (1..127) for channel ch (0..32). Use `ch = 0` to set for all channels.

I2M.NOTE

- **I2M.NOTE x y**
- *alias*: **I2M.N**

Send MIDI Note On message for note number x (0..127) with velocity y (1..127) on current channel. A velocity of `0` will be treated as a MIDI Note Off message.

I2M.NOTE.O

- **I2M.NOTE.O x**
- *alias*: **I2M.NO**

Send a manual MIDI Note Off message for note number x (0..127). This can be used either before i2c2midi sends the automatic Note Off message (to stop the note from playing before its originally planned ending), or in combination with `I2M.TIME` set to `0` (in which case i2c2midi does not send automatic Note Off messages).

I2M.NT

- **I2M.NT x y z**

Send MIDI Note On message for note number x (0..127) with velocity y (1..127) and note duration z ms (0..32767).

I2M.CC

- **I2M.CC x y**

Send MIDI CC message for controller x (0..127) with value y (0..127).

I2M.CC.SET

- **I2M.CC.SET x y**

Send MIDI CC message for controller x (0..127) with value y (0..127), bypassing any slew settings.

I2M.CCV

- **I2M.CCV x y**

Send MIDI CC message for controller x (0..127) with volt value y (0..16383, 0..+10V).

I2M.CC.OFF

- **I2M.CC.OFF x** / **I2M.CC.OFF x y**

Get current offset / Set offset of values of controller x (0..127) to y (-127..127). Default is `y = 0`.

I2M.CC.SLEW

- **I2M.CC.SLEW x** / **I2M.CC.SLEW x y**

Get current slew time for controller x / Set slew time for controller x (0..127) to y ms (0..32767). i2c2midi will ramp from the controller's last value to a new value within the given time x, sending MIDI CCs at a maximum rate of 30 ms. If the slewing is still ongoing when a new value is set, the slewing uses its current position as the last value. Is 8 CC controller values can be slewed simoultaneously before the oldest currently slewing value is overwritten by the newest. Default is `y = 0`.

I2M.NRPN

- **I2M.NRPN x y z**

Send MIDI NRPN message (high-res CC) for parameter MSB x and LSB y with value y (0..16383).

I2M.NRPN.OFF

- **I2M.NRPN.OFF x y** / **I2M.NRPN.OFF x y z**

Get current offset / Set offset of values of NRPN messages to z (-16384..16383). Default is z = 0.

I2M.NRPN.SLEW

- **I2M.NRPN.SLEW x y** / **I2M.NRPN.SLEW x y z**

Get current slew time / Set slew time for NRPN messages to z ms (0..32767). Default is z = 0.

I2M.NRPN.SET

- **I2M.NRPN.SET x y z**

Send MIDI NRPN message for parameter MSB x and LSB y with value y (0..16383), bypassing any slew settings.

I2M.CHORD

- **I2M.CHORD x y z**
- *alias*: **I2M.C**

Play chord x (1..8) with root note y (-127..127) and velocity z (1..127). A chord consists of up to eight notes defined relative to the root note via `I2M.C.ADD`, `I2M.C.RM`, `I2M.C.INS`, `I2M.C.DEL` or `I2M.C.SET`, which are sent out as MIDI Note On messages in the order they are defined in the chord. If no note has been defined in the chord yet, no note will be played. 8 chords can be defined using their respective index 1..8.

I2M.C.ADD

- **I2M.C.ADD x y**
- *alias*: **I2M.C+**

Add note y (-127..127) to chord x (0..8), with y relative to the root note specified when playing a chord. E.g. add `0`, 4 and 7 to define a major triad. Or go more experimental, e.g. −2, 13, 2, 13. Up to eight chords can be defined, with eight notes each. Use `x = 0` to add the note to all chords.

I2M.C.RM

- **I2M.C.RM x y**
- *alias*: **I2M.C-**

Remove note y (-127..127) from chord x (0..8). If the chord contains note y multiple times, the latest instance is removed. If the chord does not contain the note the message is simply ignored. Use `x = 0` to remove the note from all chords.

I2M.C.INS

- **I2M.C.INS x y z**

Add note z (-127..127) to chord x (0..8) at index y (0..7), with z relative to the root note. Already defined notes at index y and higher are pushed to the right. Use `x = 0` to insert the note to all chords.

I2M.C.DEL

- **I2M.C.DEL x y**

Delete note at index y (0..7) from chord x (0..8). Notes at index `y + 1` and higher are pushed to the left. If y is higher than the length of the chord, the message is ignored. Use `x = 0` to delete the note from all chords.

I2M.C.SET

- **I2M.C.SET x y z**

Set note at index y (0..7) in chord x (0..8) to note z (-127..127), replacing what was defined earlier at this index. If y is higher than the length of the chord, the message is ignored. Use `x = 0` to set the note in all chords.

I2M.C.B

- **I2M.C.B x y**

Clear and define chord x (0..8) using reverse binary notation (`R...`). Use `1` or `0` in order to include or exclude notes from the chord. E.g. use `x = R10001001` for `0,4,7` (major triad) or `x = R1000000100000001` for `0,7,15`. y can be a maximum of 16 digit long. Use `x = 0` to clear and define all chords.

I2M.C.CLR

- **I2M.C.CLR x**

Clear chord x (0..8). Use `x = 0` to clear all chords.

I2M.C.L

- **I2M.C.L x** / **I2M.C.L x y**

Get current length / Set length of chord x (0..8) to y (1..8). The length of a chord changes automatically each time a note is added or removed. Values of x higher than number of actual defined notes are ignored. Already defined notes are not affected by setting the chord length, but won't be played if their index is outside of the set chord length. Use `x = 0` to set the length of all chords.

I2M.C.SC

- **I2M.C.SC x y**

Set scale for chord x (0..8) based on chord y (0..8). Setting a scale for a chord comes in handy when using chord transformations that introduce new notes, like `I2M.C.TRP`, `I2M.C.DIS` or `I2M.C.REF`. Use `y = 0` to remove the scale. Use `x = 0` to set reversal for all chords.

I2M.C.REV

- **`I2M.C.REV x y`**

Set reverse of notes in chord x (0..8) to y. `y = 0` or an even number means not reversed, `y = 1` or an uneven number means reversed. E.g. `y = 1` for chord `0,3,7` will lead to `7,3,0`. Default is `y = 0`. Use `x = 0` to reverse all chords.

I2M.C.ROT

- **`I2M.C.ROT x y`**

Set rotation of notes in chord x (0..8) to y steps (-127..127). E.g. `y = 1` of chord `0,3,7` will lead to `3,7,0`, `y = 2 will lead to7,0,3`, y = -1`will lead to7,0,3. Default is`y = 0. Use`x = 0‘ to set rotation for all chords.

I2M.C.TRP

- **`I2M.C.TRP x y`**

Set transposition of chord x (0..8) to y (-127..127). Transposition adds y to the note number of each note in the chord.Default is `y = 0`. Use `x = 0` to set transposition for all chords. This transformation introduces new notes to the chord – try it in combination with setting a scale.

I2M.C.DIS

- **`I2M.C.DIS x y z`**

Set distortion of chord x (0..8) to width y (-127..127) with anchor point z (0..16). Distortion adds y+n to the note number of each note in the chord. The anchor point influences the direction and amount (n) of the transformation. Default is `y = 0`. Use `x = 0` to set distortion for all chords. This transformation introduces new notes to the chord – try it in combination with setting a scale.

I2M.C.REF

- **`I2M.C.REF x y z`**

Set reflection of chord x (0..8) to y (-127..127) with anchor point z (0..16). The anchor point defines at which axis the chord gets reflected. Default is `y = 0`. Use `x = 0` to set distortion for all chords. This transformation introduces new notes to the chord – try it in combination with setting a scale.

I2M.C.INV

- **I2M.C.INV x y**

Set inversion of chord x (0..8) to y (-32..32). Default is `y = 0`. Use `x = 0` to set inversion for all chords.

I2M.C.STR

- **I2M.C.STR x y**

Set strumming of chord x (0..8) to x ms (0..32767). Strumming plays the notes of a chord arpeggiated, with an interval of y ms in between notes. Default is `y = 0`. Use `x = 0` to set strumming for all chords.

I2M.C.VCUR

- **I2M.C.VCUR w x y z**
- *alias*: **I2M.C.V~**

Set velocity curve for chord `w` (0..8) with curve type x (0..5), start value y% (0..32767) and end value z% (0..32767). This will affect the velocity of the notes in the order they are defined in the chord. Start and end percentages refer to the velocity with which the chord is played via `I2M.C`. Use `x = 0` to turn velocity curve off. The following curves are available: 0) Off 1) Linear 2) Exponential 3) Triangle 4) Square 5) Random. Use `w = 0` to set velocity curve for all chords. Try a random curve with subtle values for a humanizing effect.

I2M.C.TCUR

- **I2M.C.TCUR w x y z**
- *alias*: **I2M.C.T~**

Set time curve for chord `w` (0..8) with curve type x (0..5), start value y% (0..32767) and end value z% (0..32767). This will affect the time interval between the notes in the order they are defined in the chord. Start and end percentages refer to the current strumming setting of the chord, set via `I2M.C.STR`. Use `x = 0` to turn time curve off. The following curves are available: 0) Off 1) Linear 2) Exponential 3) Triangle 4) Square 5) Random. Use `w = 0` to set time curve for all chords. Try a square curve with similar values to create swing. Try a random curve with subtle values for a humanizing effect.

I2M.C.DIR

- **I2M.C.DIR x y**

Set play direction for chord x (0..8) to direction y (0..8). This will affect the order in which chord notes are played. Make sure to set strumming via `I2M.C.STR`. The following directions are available: 0) Forward (0,1,2,3,4) 1) Backward (4,3,2,1,0) 2) Inside out (2,1,3,0,4) 3) Outside in (0,4,1,3,2) 4) Random (2,3,1,0,4) 5) Bottom repeat

(0,1,0,2,0,3,0,4) 6) Top repeat (0,4,1,4,2,4,3,4) 7) Pingpong (0,1,2,3,4,3,2,1,0) 8) Ping & pong (0,1,2,3,4,4,3,2,1,0). Default is `y = 0`.

I2M.C.QN

- **I2M.C.QN x y z**

Get the transformed note number of a chord note for chord x (1..8) with root note y (-127..127) at index z (0..7). The response is the absolute note number (0..127). Use this OP to send the transformed note number to other devices within eurorack, e.g. via V/OCT to any oscillator or via I2C to I2C-enabled devices like Just Friends or disting EX.

I2M.C.QV

- **I2M.C.QV x y z**

Get the transformed note velocity of a chord note for chord x (1..8) with root velocity y (1..127) at index z (0..7). The response is the absolute note velocity (0..127). Use this OP to send the transformed note velocity to other devices within eurorack, e.g. via CV to a VCA or via I2C to I2C-enabled devices like Just Friends or disting EX.

I2M.B.R

- **I2M.B.R x**

Turn recording of notes into the buffer on or off. `x = 1` is on, `x = 0` is off. If recording is turned on, all outgoing MIDI notes are recorded into the buffer, storing note number, note velocity, note duration and MIDI channel.

I2M.B.L

- **I2M.B.L x**

Set the length of the buffer to x ms (0..32767). Default is `x = 1000`.

I2M.B.START

- **I2M.B.START x**

Add an offset of x ms (0..32767) to the start of the buffer. The offset time is non-distructively added to the start of the looping buffer. E.g. if the buffer length is set to `1000` ms and start offset is set to `200` ms, the buffer will loop the section `200` - `1000` ms, resulting in a looping buffer length of `800` ms. Default is `x = 0`.

I2M.B.END

- **I2M.B.END x**

Add a negative offset of x ms (0..32767) to the end of the buffer. The offset time is non-distructively substracted from the end of the looping buffer. E.g. if the buffer length is set to 1000 ms, start offset is set to 200 ms, and end offset is set to 300 ms, the buffer will loop the section 200 - 700 ms, resulting in a looping buffer length of 500 ms. Default is x = 0.

I2M.B.DIR

- **I2M.B.DIR x**

Set the play direction x (0..2) of the buffer. The following directions are available: 0) Forward 1) Backward 2) Pingpong. Keep in mind that changing the direction only affects notes that have been already recorded to the buffer before the change in direction; all notes recorded afterwards are recorded relative to the new direction. Default is x = 0.

I2M.B.SPE

- **I2M.B.SPE x**

Set the playing speed x (1..32767) of the buffer. x = 100 is equivalent to 'normal speed', x = 50 means double the speed, x = 200 means half the speed, etc. Of course, all values in between can be chosen. Keep in mind that changing the speed only affects notes that have been already recorded to the buffer before the change in speed; all notes recorded afterwards are recorded relative to the new speed. Default is x = 100.

I2M.B.FB

- **I2M.B.FB x**

Set the feedback length x (0..255) of the buffer. By default, each recorded note is getting decreased in velocity with each buffer iteration. The feedback value determines, how many buffer iterations a recorded note will 'survive' in the buffer, before the decreasing velocity will reach zero (meaning the note is remove from the buffer). Set x = 0 to turn off the automatic decrease in velocity, keeping notes in the buffer indefinitely. Use this setting in combination with I2M.B.VSHIFT or I2M.B.CLR. Default is x = 8.

I2M.B.NSHIFT

- **I2M.B.NSHIFT x**

Set the note shift of recorded notes to x semitones (-127..127). With each buffer iteration, this value gets added accumulatively to the original note number. E.g. with a note shift setting of x = 12, a recorded note 60 will be played as note 72 during the first buffer iteration, as note 84 during the second iteration, etc. Default is x = 0.

I2M.B.VSHIFT

- **I2M.B.VSHIFT x**

Set the velocity shift of recorded notes to x (-127..127). With each buffer iteration, this value gets added accumulatively to the original note velocity. E.g. with a velocity shift setting of `x = -10`, a recorded note with velocity `110` will be played with velocity `100` during the first buffer iteration, with velocity `90` during the second iteration, etc. Default is `x = 0`. Please note: This setting is the twin sibling of `I2M.B.FB`: While `I2M.B.FB` defines the number of iterations determining the amount of change in velocity per iteration, `I2M.B.VSHIFT` defines the amount of change in velocity per iteration determining the number of iterations.

I2M.B.TSHIFT

- **I2M.B.TSHIFT x**

Set the note duration shift ('time shift') of recorded notes to x ms (-16384..16383). With each buffer iteration, this value gets added accumulatively to the original note duration. E.g. with a duration shift setting of `x = 100`, a recorded note with duration `200` will be played with duration `300` during the first buffer iteration, with duration `400` during the second iteration, etc. Default is `x = 0`.

I2M.B.NOFF

- **I2M.B.NOFF x**

Set the note offset of recorded notes to x semitones (-127..127). This value gets added once to the original note number and is then kept for all buffer iterations. E.g. with a note offset setting of `x = 7`, a recorded note `60` will be played as note `67` for all buffer iterations. Default is `x = 0`.

I2M.B.VOFF

- **I2M.B.VOFF x**

Set the velocity offset of recorded notes to x (-127..127). This value gets added once to the original note velocity and is then kept for all buffer iterations. E.g. with a velocity offset setting of `x = -50`, a recorded note with velocity `120` will be played with velocity `70` for all buffer iterations. Default is `x = 0`.

I2M.B.TOFF

- **I2M.B.TOFF x**

Set the note duration offset ('time offset') of recorded notes to x ms (-16384..16383). This value gets added once to the original note duration and is then kept for all buffer iterations. E.g. with a duration offset setting of `x = -50`, a recorded note with duration `200` will be played with duration `150` for all buffer iterations. Default is `x = 0`.

I2M.B.MODE

- **I2M.B.MODE x**

Set the buffer mode to x (0..1). The buffer can work in two different modes: 1) Digital 2) Tape. In Digital mode, the buffer speed (set via I2M.B.SPE) works independent of note number and note duration: If the buffer speed changes, the note number and note duration of a recorded note stays unaffected. In Tape mode on the other hand, the buffer speed affects the note number and note duration of recorded notes in the buffer, mimicking the behaviour of real tape. If the buffer speed gets doubled, the note number is pitched up by one octave and the note duration gets divided in half. Similarly, if the buffer speed gets divided in half, the note number is pitched down an octave and the note duration gets doubled, etc. Default is x = 0.

I2M.Q.CH

- **I2M.Q.CH** / **I2M.Q.CH x**
- *alias*: **I2M.Q.#**

Get currently set MIDI channel / Set MIDI channel x (1..16) for MIDI in. Default is x = 1.

I2M.Q.LATCH

- **I2M.Q.LATCH x**

Turn on or off 'latching' for MIDI notes received via MIDI in. x = 0 means Note Off messages are recorded in the note history, so only notes with keys currently held down on the MIDI controller are stored. x = 1 means Note Off messages are not recorded in the note history, so notes are still stored after releasing the respective key on the MIDI controller. Default is x = 1.

Advanced

Teletype terminology

Here is a picture to help understand the naming of the various parts of a Teletype command:

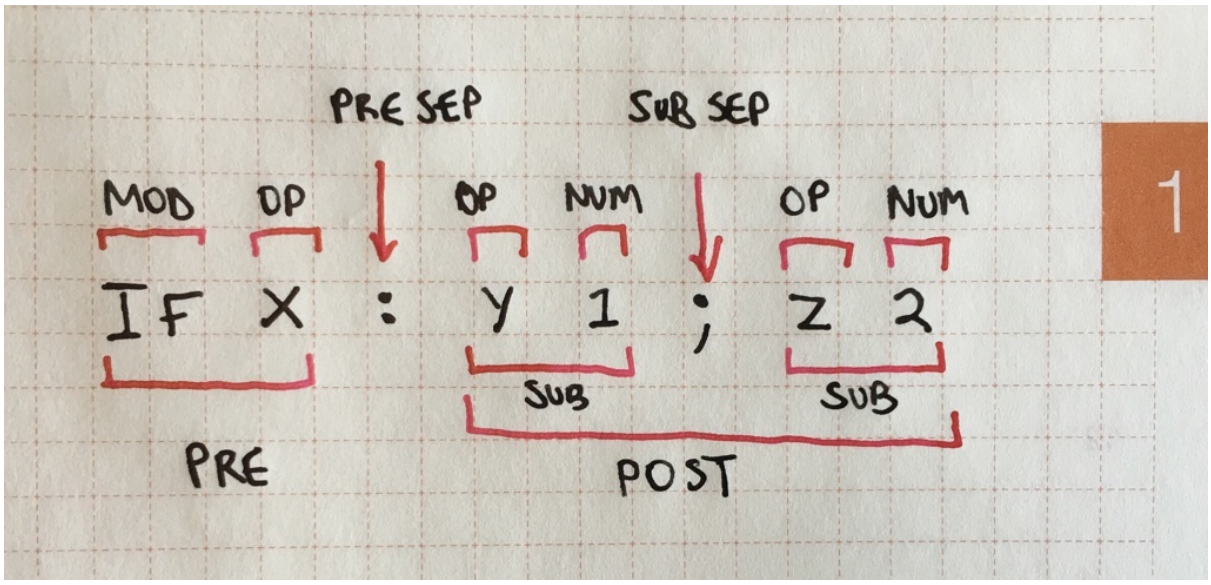

Figure 2: Teletype command terminology

COMMAND The entire command, e.g. `IF X: Y 1; Z 2;`.

PRE The (optional) part before the *PRE SEP*, e.g. `IF X`.

POST The part after the *PRE SEP*, e.g. `Y 1; Z 2`.

SUB A sub command (only allowed in the *POST*), e.g. `Y 1`, or `Z 2`.

PRE SEP A *colon*, only one is allowed.

SUB SEP A *semi-colon*, that separates sub commands (if used), only allowed in the *POST*.

NUM A number between −32768 and 32767.

OP An *operator*, e.g. X, TR.PULSE

MOD A *modifier*, e.g. IF, or L.

Sub commands

Sub commands allow you to run multiple commands on a single line by utilising a semi-colon to separate each command, for example the following script:

```
X 0
Y 1
Z 2
```

Can be rewritten using sub commands as:

```
X 0; Y 1; Z 2
```

On their own sub commands allow for an increased command density on the Teletype. However when combined with PRE statements, certain operations become a lot easier.

Firstly, sub commands cannot be used before a `MOD` or in the PRE itself. For example, the following is **not allowed**:

```
X 1; IF X: TR.PULSE 1
```

We can use them in the POST though, particularly with an `IF`, for example:

```
IF X: CV 1 N 60; TR.P 1
IF Y: TR.P 1; TR.P 2; TR.P 3
```

Sub commands can also be used with `L`.

Aliases

In general, aliases are a simple concept to understand. Certain OPs have been given shorted names to save space and the amount of typing, for example:

```
TR.PULSE 1
```

Can be replaced with:

```
TR.P 1
```

Where confusion may arise is with the symbolic aliases that have been given to some of the maths OPs. For instance **+** is given as an alias for `ADD` and it *must* be used as a direct replacement:

```
X ADD 1 1
X + 1 1
```

The key to understanding this is that the Teletype uses *prefix notation*[36] always, even when using mathematical symbols.

The following example (using *infix notation*) **will not work**:

```
X 1 + 1
```

[36] Also know as *Polish notation*.

Aliases are entirely optional, most OPs do not have aliases. Consult the OP tables and documentation to find them.

Avoiding non-determinism

Although happy accidents in the modular world are one of it's many joys, when writing computer programs they can be incredibly frustrating. Here are some small tips to help keep things predictable (when you want them to be):

1. **Don't use variables unless you need to.**

   This is not to say that variables are not useful, rather it's the opposite and they are extremely powerful. But it can be hard to keep a track of what each variable is used for and on which script it is used. Rather, try to save using variables for when you do want non-deterministic (i.e. *variable*) behaviour.

2. **Consider using `I` as a temporary variable.**

   If you do find yourself needing a variable, particularly one that is used to continue a calculation on another line, consider using the variable `I`. Unlike the other variables, `I` is overwritten whenever `L` is used, and as such, is implicitly transient in nature. One should never need to worry about modifying the value of `I` and causing another script to malfunction, as no script should ever assume the value of `I`.

3. **Use PN versions of OPs.**

   Most `P` OPs are now available as `PN` versions that ignore the value of `P.I`. (e.g. `PN.START` for `P.START`). Unless you explicitly require the non-determinism of P versions, stick to the PN versions (space allowing).

4. **Avoid using A, B, C and D to refer to the trigger outputs, instead use the numerical values directly.**

   As `A-D` are variables, they may no longer contain the values `1-4`, and while this was the recommend way to name triggers, it is no longer consider ideal. Newer versions of the Teletype hardware have replaced the labels on the trigger outputs, with the numbers 1 to 4.

Grid integration

Grid integration can be described very simply: it allows you to use grid with teletype. However, there is more to it than just that. You can create custom grid interfaces that can be tailored individually for each scene. Since it's done with scripts you can dynamically change these interfaces at any point - you could even create a dynamic interface that reacts to the scene itself or incoming triggers or control voltages.

You can simply use grid as an LED display to visualize your scene. Or make it into an earthsea style keyboard. You can create sequencers, or control surfaces to control

other sequencers. The grid operators simplify building very complex interfaces, while something simple like a bank of faders can be done with just two lines of scripts.

Grid integration consists of 3 main features: grid operators, Grid Visualizer, and Grid Control mode. Grid operators allow you to draw on grid or create grid controls, such as buttons and faders, that can trigger scripts when pressed. As with any other operators you can execute them in Live screen or use them in any of your scripts.

Grid Visualizer provides a virtual grid within the Teletype itself:

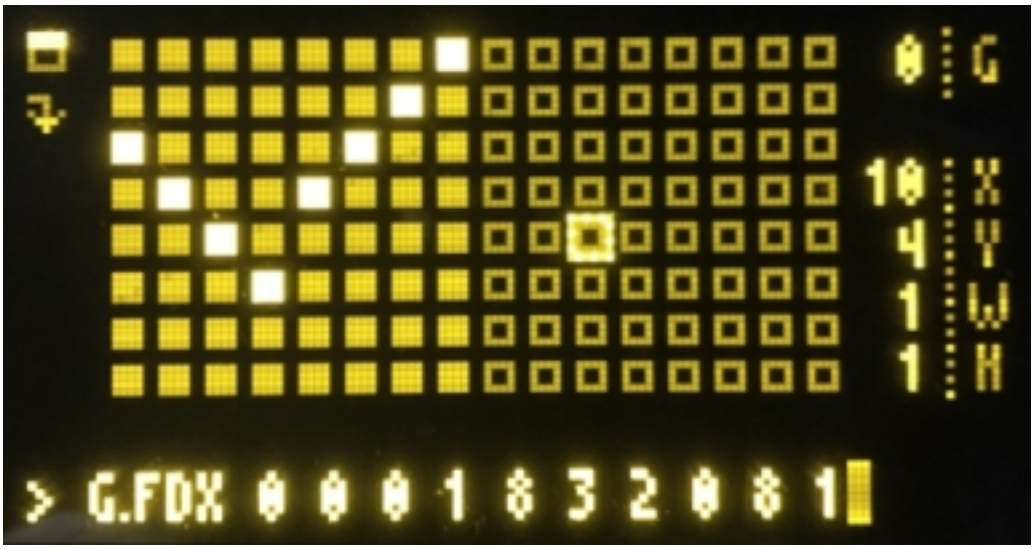

Figure 3: Grid Visualizer

It can be very useful while developing a script as you don't have to switch between the grid and the keyboard as often. To turn it on navigate to Live screen and press `Alt-G` (press again to switch to Full View / turn it off). You can also emulate button presses, which means it can even be used as an alternative to grid if you don't have one, especially in full mode - try it with one of the many grid scenes[37] already developed. For more information on how to use it please refer to the Grid Visualizer documentation[38].

Grid Control Mode is a built in grid interface that allows you to use grid to trigger and mute scripts, edit variables and tracker values, save and load scenes, and more. It's available in addition to whatever grid interface you develop - simply press the front panel button while the grid is attached. It can serve as a simple way to use grid to control any scene even without using grid ops, but it can also be very helpful when used together with a scripted grid interface. For more information and diagrams please refer to the Grid Control documentation[39],

If you do want to try and build your own grid interfaces the Grid Studies[40] is the best place to start.

[37] https://github.com/scanner-darkly/teletype/wiki/CODE-EXCHANGE
[38] https://github.com/scanner-darkly/teletype/wiki/GRID-VISUALIZER
[39] https://github.com/scanner-darkly/teletype/wiki/GRID-CONTROL-MODE
[40] https://github.com/scanner-darkly/teletype/wiki/GRID-INTEGRATION

Alphabetical list of OPs and MODs

OP	OP *(set)*	*(aliases)*	Description
$F script			execute script as a function
$F1 script param			execute script as a function with 1 parameter
$F2 script param1 param2			execute script as a function with 2 parameters
$L script line			execute script line
$L1 script line param			execute script line as a function with 1 parameter
$L2 script line param1 param2			execute script line as a function with 2 parameters
$S line			execute script line within the same script as a function
$S1 line param			execute script line within the same script as a function with 1 parameter
$S2 line param1 param2			execute script line within the same script as a function with 2 parameters
& x y			bitwise and x & y
? x y z			if condition x is true return y, otherwise return z
@	**@ x**		get or set the current pattern value under the turtle
@BOUNCE	**@BOUNCE 1**		get whether the turtle fence mode is BOUNCE, or set it to BOUNCE with 1
@BUMP	**@BUMP 1**		get whether the turtle fence mode is BUMP, or set it to BUMP with 1
@DIR	**@DIR x**		get the direction of the turtle's @STEP in degrees or set it to x
@F x1 y1 x2 y2			set the turtle's fence to corners x1,y1 and x2,y2
@FX1	**@FX1 x**		get the left fence line or set it to x
@FX2	**@FX2 x**		get the right fence line or set it to x

OP	OP *(set)*	*(aliases)*	Description
@FY1	**@FY1 x**		get the top fence line or set it to x
@FY2	**@FY2 x**		get the bottom fence line or set it to x
@MOVE x y			move the turtle x cells in the X axis and y cells in the Y axis
@SCRIPT	**@SCRIPT x**		get which script runs when the turtle changes cells, or set it to x
@SHOW	**@SHOW 0/1**		get whether the turtle is displayed on the TRACKER screen, or turn it on or off
@SPEED	**@SPEED x**		get the speed of the turtle's @STEP in cells per step or set it to x
@STEP			move @SPEED/100 cells forward in @DIR, triggering @SCRIPT on cell change
@WRAP	**@WRAP 1**		get whether the turtle fence mode is WRAP, or set it to WRAP with 1
@X	**@X x**		get the turtle X coordinate, or set it to x
@Y	**@Y x**		get the turtle Y coordinate, or set it to x
A	**A x**		get / set the variable A, default 1
ABS x			absolute value of x
ADD x y		**+**	add x and y together
AND x y		**&&**	logical AND of x and y
AND3 x y z		**&&&**	logical AND of x, y and z
AND4 x y z a		**&&&&**	logical AND of x, y, z and a
ANS.A	**ANS.A n d**		send arc encoder event for ring n, delta d
ANS.A.LED n x			read arc LED buffer for ring n, LED x clockwise from north
ANS.APP	**ANS.APP x**		get/set active app
ANS.G x y	**ANS.G x y z**		get/set grid key on/off state (z) at position x, y
ANS.G.LED x y			get grid LED buffer at position x, y
ANS.G.P x y			simulate grid key press at position (x, y)
ARP.DIV v d			set voice clock divisor (euclidean length), range [1-32]

OP	OP *(set)*	*(aliases)*	Description
ARP.ER v f d r			set all euclidean rhythm
ARP.FIL v f			set voice euclidean fill, use 1 for straight clock division, range [1-32]
ARP.GT v g			set voice gate length [0-127], scaled/synced to course divisions of voice clock
ARP.HLD h			0 disables key hold mode, other values enable
ARP.RES v			reset voice clock/pattern on next base clock tick
ARP.ROT v r			set voice euclidean rotation, range [-32, 32]
ARP.RPT v n s			set voice pattern repeat, n times [0-8], shifted by s semitones [-24, 24]
ARP.SHIFT v o			shift voice cv by standard tt pitch value (e.g. N 6, V -1, etc)
ARP.SLEW v t			set voice slew time in ms
ARP.STY y			set base arp style [0-7]
AVG x y			the average of x and y
B	**B x**		get / set the variable B, default 2
BCLR x y			clear bit y in value x
BGET x y			get bit y in value x
BPM x			milliseconds per beat in BPM x
BREAK		**BRK**	halts execution of the current script
BREV x			reverse bit order in value x
BSET x y			set bit y in value x
BTOG x y			toggle bit y in value x
C	**C x**		get / set the variable C, default 3
CHAOS x			get next value from chaos generator, or set the current value
CHAOS.ALG x			get or set the algorithm for the CHAOS generator. 0 = LOGISTIC, 1 = CUBIC, 2 = HENON, 3 = CELLULAR
CHAOS.R x			get or set the R parameter for the CHAOS generator
CROW.AR x y z t			Creates an envelope on output x, rising in y ms, falling in z ms, and reaching height t.

OP	OP *(set)*	*(aliases)*	Description
CROW.C1 x			Calls the function `ii.self.call1(x)` on crow.
CROW.C2 x y			Calls the function `ii.self.call2(x, y)` on crow.
CROW.C3 x y z			Calls the function `ii.self.call3(x, y, z)` on crow.
CROW.C4 x y z t			Calls the function `ii.self.call4(x, y, z, t)` on crow.
CROW.IN x			Gets voltage at input x.
CROW.LFO x y z t			Starts an envelope on output x at rate y where `0 = 1Hz` with 1v/octave scaling. z sets amplitude and `t` sets skew for assymetrical triangle waves.
CROW.OUT x			Gets voltage of output x.
CROW.PULSE x y z t			Creates a trigger pulse on output x with duration y (ms) to voltage z with polarity `t`.
CROW.Q0			Returns the result of calling the function `crow.self.query0()`.
CROW.Q1 x			Returns the result of calling the function `crow.self.query1(x)`.
CROW.Q2 x y			Returns the result of calling the function `crow.self.query2(x, y)`.
CROW.Q3 x y z			Returns the result of calling the function `crow.self.query3(x, y, z)`.
CROW.RST			Calls the function `crow.reset()` returning crow to default state.
CROW.SEL x			Sets target crow unit (1 (default), to 4).
CROW.SLEW x y			Sets output x slew rate to y milliseconds.
CROW.V x y			Sets output x to value y. Use V y for volts.

OP	OP *(set)*	*(aliases)*	Description
CROW1: ...			Send following CROW OPs to unit 1 ignoring the currently selected unit.
CROW2: ...			Send following CROW OPs to unit 2 ignoring the currently selected unit.
CROW3: ...			Send following CROW OPs to unit 3 ignoring the currently selected unit.
CROW4: ...			Send following CROW OPs to unit 4 ignoring the currently selected unit.
CROWN: ...			Send following CROW OPs to all units starting with selected unit.
CV x	**CV x y**		CV target value
CV.CAL n mv1v mv3v			Calibrate CV output n
CV.CAL.RESET n			Reset calibration data for CV output n
CV.GET x			Get current CV value
CV.OFF x	**CV.OFF x y**		CV offset added to output
CV.SET x y			Set CV value, ignoring slew
CV.SLEW x	**CV.SLEW x y**		Get/set the CV slew time in ms
CY.CV x			get the current CV value for channel x
CY.POS x	**CY.POS x y**		get / set position of channel x (x = `0` to set all), position between `0`-`255`
CY.PRE	**CY.PRE x**		return current preset / load preset x
CY.RES x			reset channel x (`0` = all)
CY.REV x			reverse channel x (`0` = all)
D	**D x**		get / set the variable `D`, default 4
DEL x: ...			Delay command by x ms
DEL.B delay_time bitmask: ...			Trigger the command up to 16 times at intervals of `delay_time` ms. Active intervals set in 16-bit `bitmask`, LSB = immediate.
DEL.CLR			Clear the delay buffer

OP	OP *(set)*	*(aliases)*	Description
DEL.G x delay_time num denom: ...			Trigger the command once immediately and `x - 1` times at ms intervals of `delay_time * (num/denom)^n` where n ranges from 0 to `x - 1`.
DEL.R x delay_time: ...			Trigger the command following the colon once immediately, and delay `x - 1` commands at `delay_time` ms intervals
DEL.X x delay_time: ...			Delay x commands at `delay_time` ms intervals
DEVICE.FLIP			Flip the screen/inputs/outputs
DIV x y		**/**	divide x by y
DR.P b p s			Drum pattern helper, b is the drum bank (`0-4`), p is the pattern (0-215) and step is the step number (0-15), returns `0` or `1`
DR.T b p q l s			Tresillo helper, b is the drum bank (`0-4`), p is first pattern (0-215), q is the second pattern (0-215), `l` is length (`1-64`), and step is the step number (0-length-1), returns `0` or `1`
DR.V p s			Velocity helper. p is the pattern (0-19). s is the step number (0-15)
DRUNK	**DRUNK x**		changes by `-1`, `0`, or `1` upon each read saving its state, setting will give it a new value for the next read
DRUNK.MAX	**DRUNK.MAX x**		set the upper bound for DRUNK, default `255`
DRUNK.MIN	**DRUNK.MIN x**		set the lower bound for DRUNK, default `0`
DRUNK.SEED	**DRUNK.SEED x**	**DRUNK.SD**	get / set the random number generator seed for the DRUNK op
DRUNK.WRAP	**DRUNK.WRAP x**		should DRUNK wrap around when it reaches it's bounds, default `0`
ELIF x: ...			if all previous `IF` / `ELIF` fail, and x is not zero, execute command
ELSE: ...			if all previous `IF` / `ELIF` fail, excute command
EQ x y		**==**	does x equal y

OP	OP *(set)*	*(aliases)*	Description
ER f l i			Euclidean rhythm, `f` is fill (1–32), `l` is length (1–32) and `i` is step (any value), returns `0` or 1
ES.CLOCK x			If II clocked, next pattern event
ES.CV x			get the current CV value for channel x
ES.MAGIC x			Magic shape (1= halfspeed, 2=doublespeed, 3=linearize)
ES.MODE x			Set pattern clock mode. (0=normal, 1=II clock)
ES.PATTERN x			Select playing pattern (0-15)
ES.PRESET x			Recall preset x (0-7)
ES.RESET x			Reset pattern to start (and start playing)
ES.STOP x			Stop pattern playback.
ES.TRANS x			Transpose the current pattern
ES.TRIPLE x			Recall triple shape (1-4)
EVERY x: ...		**EV**	run the command every x times the command is called
EX	**EX x**		get or set currently selected unit to x (1-4)
EX.A1	**EX.A1 x**		get or set the left dual algorithm
EX.A12 x y			set both dual algorithms
EX.A2	**EX.A2 x**		get or set the right dual algorithm
EX.AL.CLK			send clock to Augustus Loop
EX.AL.P x			set Augustus Loop pitch to value x
EX.ALG	**EX.ALG x**	**EX.A**	get or set the current algorithm to x (single algorithms only)
EX.ALLOFF		**EX.AO**	all notes off
EX.CH x		**EX.#**	select default note channel (for multi channel algorithms like Poly FM)
EX.CTRL x y		**EX.C**	set I2C controller x to value y
EX.LP x			get current state for loop x
EX.LP.CLR x			clear loop x
EX.LP.DOWN x			toggle octave down for loop x
EX.LP.DOWN? x			return 1 if loop x is transposed octave down, 0 otherwise
EX.LP.PLAY x			toggle playback for loop x
EX.LP.REC x			toggle recording for loop x

OP	OP *(set)*	*(aliases)*	Description
EX.LP.REV x			toggle reverse for loop x
EX.LP.REV? x			returns 1 if loop x is reversed, 0 otherwise
EX.M.CC x y			send MIDI CC message for controller x (0..127) and value y (0..127)
EX.M.CC# x y z			send MIDI CC message on channel x for controller y (0..127) and value z (0..127)
EX.M.CH	**EX.M.CH x**		get or set the currently selected MIDI channel (1-16)
EX.M.CLK			send MIDI clock message
EX.M.CONT			send MIDI Continue message
EX.M.N x y			send MIDI Note On message for note x (0..127) and velocity y (0..127)
EX.M.N# x y z			send MIDI Note On message on channel x for note y (0..127) and velocity z (0..127)
EX.M.NO x			send MIDI Note off message for note x (0..127)
EX.M.NO# x y			send MIDI Note off message on channel x for note y (0..127)
EX.M.PB x			send MIDI Pitchbend message
EX.M.PRG x			send MIDI Program Change message
EX.M.START			send MIDI Start message
EX.M.STOP			send MIDI Stop message
EX.MAX x			get the maximum possible value for parameter x
EX.MAX1 x			get left algorithm parameter maximum value
EX.MAX2 x			get right algorithm parameter maximum value
EX.MIN x			get the minimum possible value for parameter x
EX.MIN1 x			get left algorithm parameter minimum value
EX.MIN2 x			get right algorithm parameter minimum value
EX.N# x y z			send a note to channel x using pitch y and velocity z (voice allocated by the Disting)

OP	OP *(set)*	*(aliases)*	Description
EX.NO# x y			send a note off to channel x using pitch y
EX.NOTE x y		**EX.N**	send a note using pitch x and velocity y (voice allocated by the Disting)
EX.NOTE.O x		**EX.NO**	send a note off using pitch x
EX.P1 x	**EX.P1 x y**		get left algorithm parameter x or set it to value y
EX.P2 x	**EX.P2 x y**		get right algorithm parameter x or set it to value y
EX.PARAM x	**EX.PARAM x y**	**EX.P**	set parameter x to value y or get the current parameter value
EX.PLAY x			control WAV recorder playback: 1 to start, 0 to stop
EX.PRE1 x			load left preset from x slot
EX.PRE2 x			load right preset from x slot
EX.PRESET	**EX.PRESET x**	**EX.PRE**	load preset x or get the currently loaded preset
EX.PV x y			set parameter x using a value determined by scaling y from 0..16384 range.
EX.PV1 x y			set left algorithm parameter x using a value determined by scaling y from 0..16384 range
EX.PV2 x y			set right algorithm parameter x using a value determined by scaling y from 0..16384 range
EX.REC x			control WAV recorder recording: 1 to start, 0 to stop
EX.RESET			reset the currently loaded preset
EX.SAVE x			save to preset x
EX.SAVE1 x			save left preset to x slot
EX.SAVE2 x			save right preset to x slot
EX.SB.CC x y			send Select Bus CC message for controller x (0..127) and value y (0..127)
EX.SB.CH	**EX.SB.CH x**		get or set the currently selected Select Bus channel (1-16)
EX.SB.CLK			send Select Bus clock message
EX.SB.CONT			send Select Bus Continue message
EX.SB.N x y			send Select Bus Note On message for note x (0..127) and velocity y (0..127)

OP	OP *(set)*	*(aliases)*	Description
EX.SB.NO x			send Select Bus Note off message for note x (0..127)
EX.SB.PB x			send Select Bus Pitchbend message
EX.SB.PRG x			send Select Bus Program Change message
EX.SB.START			send Select Bus Start message
EX.SB.STOP			send Select Bus Stop message
EX.T x			send a trigger to voice x with medium velocity (use with SD Triggers algo)
EX.TV x y			send a trigger to voice x using velocity y (use with SD Triggers algo)
EX.VOX x y z		**EX.V**	send a note to voice x using pitch y and velocity z
EX.VOX.O x		**EX.VO**	send a note off to voice x
EX.VOX.P x y		**EX.VP**	set voice x to pitch y
EX.Z1	**EX.Z1 x**		get left Z knob value or set left Z parameter (0..127 range)
EX.Z2	**EX.Z2 x**		get right Z knob value or set right Z parameter (0..127 range)
EX.Z01			restore control for left Z knob and input
EX.Z02			restore control for right Z knob and input
EX1: ...			send following Disting ops to unit 1 ignoring the currently selected unit
EX2: ...			send following Disting ops to unit 2 ignoring the currently selected unit
EX3: ...			send following Disting ops to unit 3 ignoring the currently selected unit
EX4: ...			send following Disting ops to unit 4 ignoring the currently selected unit
EXP x			exponentiation table lookup. 0-16383 range (V 0-10)
EZ x		**!**	x is 0, equivalent to logical NOT

OP	OP *(set)*	*(aliases)*	Description
FADER x		**FB**	Reads the value of the FADER slider x; default return range is from 0 to 16383. Up to four Faderbanks can be addressed; x value between 1 and 16 correspond to Faderbank 1, x between 17 and 32 to Faderbank 2, etc…
FADER.CAL.MAX x		**FB.C.MAX**	Reads FADER x maximum position and assigns the maximum point
FADER.CAL.MIN x		**FB.C.MIN**	Reads FADER x minimum position and assigns a zero value
FADER.CAL.RESET x		**FB.C.R**	Resets the calibration for FADER x
FADER.SCALE x y z		**FB.S**	Set static scaling of the FADER x to between min and max.
FLIP	**FLIP x**		returns the opposite of its previous state (0 or 1) on each read (also settable)
FR	**FR x**		get/set the return value when a script is called as a function
G.BTN id x y w h type level script			initialize button
G.BTN.EN id	**G.BTN.EN id x**		enable/disable button or check if enabled
G.BTN.L id	**G.BTN.L id level**		get/set button level
G.BTN.PR id action			emulate button press/release
G.BTN.SW id			switch button
G.BTN.V id	**G.BTN.V id value**		get/set button value
G.BTN.X id	**G.BTN.X id x**		get/set button x coordinate
G.BTN.Y id	**G.BTN.Y id y**		get/set button y coordinate
G.BTNI			id of last pressed button
G.BTNL	**G.BTNL level**		get/set level of last pressed button
G.BTNV	**G.BTNV value**		get/set value of last pressed button
G.BTNX	**G.BTNX x**		get/set x of last pressed button
G.BTNY	**G.BTNY y**		get/set y of last pressed button

OP	OP *(set)*	*(aliases)*	Description
G.BTX id x y w h type level script columns rows			initialize multiple buttons
G.CLR			clear all LEDs
G.DIM level			set dim level
G.FDR id x y w h type level script			initialize fader
G.FDR.EN id	**G.FDR.EN id x**		enable/disable fader or check if enabled
G.FDR.L id	**G.FDR.L id level**		get/set fader level
G.FDR.N id	**G.FDR.N id value**		get/set fader value
G.FDR.PR id value			emulate fader press
G.FDR.V id	**G.FDR.V id value**		get/set scaled fader value
G.FDR.X id	**G.FDR.X id x**		get/set fader x coordinate
G.FDR.Y id	**G.FDR.Y id y**		get/set fader y coordinate
G.FDRI			id of last pressed fader
G.FDRL	**G.FDRL level**		get/set level of last pressed fader
G.FDRN	**G.FDRN value**		get/set value of last pressed fader
G.FDRV	**G.FDRV value**		get/set scaled value of last pressed fader
G.FDRX	**G.FDRX x**		get/set x of last pressed fader
G.FDRY	**G.FDRY y**		get/set y of last pressed fader
G.FDX id x y w h type level script columns rows			initialize multiple faders
G.GBT group id x y w h type level script			initialize button in group
G.GBTN.C group			get count of currently pressed
G.GBTN.H group			get button block height
G.GBTN.I group index			get id of pressed button
G.GBTN.L group odd_level even_level			set level for group buttons

OP	OP *(set)*	*(aliases)*	Description
G.GBTN.V group value			set value for group buttons
G.GBTN.W group			get button block width
G.GBTN.X1 group			get leftmost pressed x
G.GBTN.X2 group			get rightmost pressed x
G.GBTN.Y1 group			get highest pressed y
G.GBTN.Y2 group			get lowest pressed y
G.GBX group id x y w h type level script columns rows			initialize multiple buttons in group
G.GFD grp id x y w h type level script			initialize fader in group
G.GFDR.L group odd_level even_level			set level for group faders
G.GFDR.N group value			set value for group faders
G.GFDR.RN group min max			set range for group faders
G.GFDR.V group value			set scaled value for group faders
G.GFX group id x y w h type level script columns rows			initialize multiple faders in group
G.GRP	**G.GRP id**		get/set current group
G.GRP.EN id	**G.GRP.EN id x**		enable/disable group or check if enabled
G.GRP.RST id			reset all group controls
G.GRP.SC id	**G.GRP.SC id script**		get/set group script
G.GRP.SW id			switch groups
G.GRPI			get last group
G.KEY x y action			emulate grid press
G.LED x y	**G.LED x y level**		get/set LED
G.LED.C x y			clear LED
G.RCT x1 y1 x2 y2 fill border			draw rectangle
G.REC x y w h fill border			draw rectangle

OP	OP *(set)*	*(aliases)*	Description
G.ROTATE x			set grid rotation
G.RST			full grid reset
GT x y		**>**	x is greater than y
GTE x y		**>=**	x is greater than or equal to y
HZ x			converts 1V/OCT value x to Hz/Volt value, useful for controlling non-euro synths like Korg MS-20
I	**I x**		get / set the variable `I`
I1			get the first parameter when executing a script as a function
I2			get the second parameter when executing a script as a function
I2M.AT x			Send MIDI After Touch message with value x (0..127)
I2M.B.CLR			Clear the buffer, erasing all recorded notes in the buffer
I2M.B.DIR x			Set the play direction x (0..2) of the buffer
I2M.B.END x			Add a negative offset of x ms (0..32767) to the end of the buffer
I2M.B.FB x			Set the feedback length x (0..255) of the buffer
I2M.B.L x			Set the length of the buffer to x ms (0..32767)
I2M.B.MODE x			Set the buffer mode to x (0..1). 1) Digital 2) Tape
I2M.B.NOFF x			Set the note offset of recorded notes to x semitones (-127..127)
I2M.B.NSHIFT x			Set the note shift of recorded notes to x semitones (-127..127)
I2M.B.R x			Turn recording of notes into the buffer on or off
I2M.B.SPE x			Set the playing speed x (1..32767) of the buffer. `x = 100` is equivalent to 'normal speed', `x = 50` means double the speed, `x = 200` means half the speed, etc.
I2M.B.START x			Add an offset of x ms (0..32767) to the start of the buffer

OP	OP *(set)*	*(aliases)*	Description
I2M.B.TOFF x			Set the note duration offset ('time offset') of recorded notes to x ms (-16384..16383)
I2M.B.TSHIFT x			Set the note duration shift ('time shift') of recorded notes to x ms (-16384..16383)
I2M.B.VOFF x			Set the velocity offset of recorded notes to x (-127..127)
I2M.B.VSHIFT x			Set the velocity shift of recorded notes to x (-127..127)
I2M.C# ch x y z			Play chord x (1..8) with root note y (-127..127) and velocity z (1..127) on channel ch (1..32)
I2M.C.ADD x y		**I2M.C+**	Add relative note y (-127..127) to chord x (0..8), use `x = 0` to add to all chords
I2M.C.B x y			Clear and define chord x (0..8) using reverse binary notation (`R...`)
I2M.C.CLR x			Clear chord x (0..8), use `x = 0` to clear all chords
I2M.C.DEL x y			Delete note at index y (0..7) from chord x (0..8), use `x = 0` to delete from all chords
I2M.C.DIR x y			Set play direction for chord x (0..8) to direction y (0..8)
I2M.C.DIS x y z			Set distortion of chord x (0..8) to y (-127..127) with anchor point z (0..16), use `x = 0` to set for all chords
I2M.C.INS x y z			Add note z (-127..127) to chord x (0..8) at index y (0..7), with z relative to the root note; use `x = 0` to insert into all chords
I2M.C.INV x y			Set inversion of chord x (0..8) to y (-32..32), use `x = 0` to set for all chords
I2M.C.L x	**I2M.C.L x y**		Get current length / Set length of chord x (0..8) to y (1..8), use `x = 0` to set length of all chords
I2M.C.QN x y z			Get the transformed note number of a chord note for chord x (1..8) with root note y (-127..127) at index z (0..7)

OP	OP *(set)*	*(aliases)*	Description
I2M.C.QV x y z			Get the transformed note velocity of a chord note for chord x (1..8) with root velocity y (1..127) at index z (0..7)
I2M.C.REF x y z			Set reflection of chord x (0..8) to y iterations (-127..127) with anchor point z (0..16), use x = 0 to set for all chords
I2M.C.REV x y			Set reversal of notes in chord x (0..8) to y. y = 0 or an even number means not reversed, y = 1 or an uneven number means reversed. Use x = 0 to set for all chords.
I2M.C.RM x y		**I2M.C-**	Remove note y (-127..127) from chord x (0..8), use x = 0 to remove from all chords
I2M.C.ROT x y			Set rotation of notes in chord x (0..8) to y steps (-127..127), use x = 0 to set for all chords
I2M.C.SC x y			Set scale for chord x (0..8) based on chord y (0..8), use x = 0 to set for all chords, use y = 0 to remove scale
I2M.C.SET x y z			Set note at index y (0..7) in chord x (0..8) to note z (-127..127), use x = 0 to set in all chords
I2M.C.STR x y			Set strumming of chord x (0..8) to x ms (0..32767), use x = 0 to set for all chords
I2M.C.TCUR w x y z		**I2M.C.T~**	Set time curve to strumming for chord w (0..8) with curve type x (0..5), start value y% (0..32767) and end value z% (0..32767), use w = 0 to set for all chords, use x = 0 to turn off
I2M.C.TRP x y			Set transposition of chord x (0..8) to y (-127..127), use x = 0 to set for all chords
I2M.C.VCUR w x y z		**I2M.C.V~**	Set velocity curve for chord w (0..8) with curve type x (0..5), start value y% (0..32767) and end value z% (0..32767), use w = 0 to set for all chords, use x = 0 to turn off

OP	OP *(set)*	*(aliases)*	Description
I2M.CC x y			Send MIDI CC message for controller x (0..127) with value y (0..127)
I2M.CC# ch x y			Send MIDI CC message for controller x (0..127) with value y (0..127) on channel ch (1..32)
I2M.CC.OFF x	**I2M.CC.OFF x y**		Get current offset / Set offset of values of controller x (0..127) to y (-127..127)
I2M.CC.OFF# ch x	**I2M.CC.OFF# ch x y**		Get current offset / Set offset of values of controller x (0..127) to y (-127..127) for channel ch (1..32)
I2M.CC.SET x y			Send MIDI CC message for controller x (0..127) with value y (0..127), bypassing any slew settings
I2M.CC.SET# ch x y			Send MIDI CC message for controller x (0..127) with value y (0..127) on channel ch (1..32), bypassing any slew settings
I2M.CC.SLEW x	**I2M.CC.SLEW x y**		Get current slew time for controller x / Set slew time for controller x (0..127) to y ms (0..32767)
I2M.CC.SLEW# ch x	**I2M.CC.SLEW# ch x y**		Get current slew time for controller x / Set slew time for controller x (0..127) to y ms (0..32767) for channel ch (1..32)
I2M.CCV x y			Send MIDI CC message for controller x (0..127) with volt value y (0..16383, 0..+10V)
I2M.CCV# ch x y			Send MIDI CC message for controller x (0..127) with volt value y (0..16383, 0..+10V) on channel ch (1..32)
I2M.CH	**I2M.CH x**	**I2M.#**	Get currently set MIDI channel / Set MIDI channel x (1..16 for TRS, 17..32 for USB) for MIDI out
I2M.CHORD x y z		**I2M.C**	Play chord x (1..8) with root note y (-127..127) and velocity z (1..127)
I2M.CLK			Send MIDI Clock message, this still needs improvement …
I2M.CONT			Send MIDI Clock Continue message

OP	OP *(set)*	*(aliases)*	Description
I2M.MAX x y			Set maximum note number for MIDI notes to x (0..127), using mode y (0..3), for current channel
I2M.MAX# ch x y			Set maximum note number for MIDI notes to x (0..127), using mode y (0..3), for channel ch (0..32)
I2M.MIN x y			Set minimum note number for MIDI notes to x (0..127), using mode y (0..3), for current channel
I2M.MIN# ch x y			Set minimum note number for MIDI notes to x (0..127), using mode y (0..3), for channel ch (0..32)
I2M.MUTE	**I2M.MUTE x**		Get mute state / Set mute state of current MIDI channel to x (0..1)
I2M.MUTE#	**I2M.MUTE# ch x**		Get mute state / Set mute state of MIDI channel ch to x (0..1)
I2M.N# ch x y			Send MIDI Note On message for note number x (0..127) with velocity y (1..127) on channel ch (1..32)
I2M.NO# ch x			Send a manual MIDI Note Off message for note number x (0..127) on channel ch (1..32)
I2M.NOTE x y		**I2M.N**	Send MIDI Note On message for note number x (0..127) with velocity y (1..127) on current channel
I2M.NOTE.O x		**I2M.NO**	Send a manual MIDI Note Off message for note number x (0..127)
I2M.NRPN x y z			Send MIDI NRPN message (high-res CC) for parameter MSB x and LSB y with value y (0..16383)
I2M.NRPN# ch x y z			Send MIDI NRPN message (high-res CC) for parameter MSB x and LSB y with value y (0..16383) on channel ch (1..32)
I2M.NRPN.OFF x y	**I2M.NRPN.OFF x y z**		Get current offset / Set offset of values of NRPN messages to z (-16384..16383)

OP	OP *(set)*	*(aliases)*	Description
I2M.NRPN.OFF# ch x y	**I2M.NRPN.OFF# ch x y z**		Get current offset / Set offset of values of NRPN messages to z (-16384..16383) for channel ch (1..32)
I2M.NRPN.SET x y z			Send MIDI NRPN message for parameter MSB x and LSB y with value y (0..16383), bypassing any slew settings
I2M.NRPN.SET# ch x y z			Send MIDI NRPN message for parameter MSB x and LSB y with value y (0..16383) on channel ch (1..32), bypassing any slew settings
I2M.NRPN.SLEW x y	**I2M.NRPN.SLEW x y z**		Get current slew time / Set slew time for NRPN messages to z ms (0..32767)
I2M.NRPN.SLEW# ch x y	**I2M.NRPN.SLEW# ch x y z**		Get current slew time / Set slew time for NRPN messages to z ms (0..32767) for channel ch (1..32)
I2M.NT x y z			Send MIDI Note On message for note number x (0..127) with velocity y (1..127) and note duration z ms (0..32767)
I2M.NT# ch x y z			Send MIDI Note On message for note number x (0..127) with velocity y (1..127) and note duration z ms (0..32767) on channel ch (1..32)
I2M.PANIC			Send MIDI Note Off messages for all notes on all channels, and reset note duration, shift, repetition, ratcheting, min/max
I2M.PB x			Send MIDI Pitch Bend message with value x (-8192..8191)
I2M.PRG x			Send MIDI Program Change message for program x (0..127)
I2M.Q.CC x			Get current value (0..127) of controller x (0..127) received via MIDI in
I2M.Q.CH	**I2M.Q.CH x**	**I2M.Q.#**	Get currently set MIDI channel / Set MIDI channel x (1..16) for MIDI in
I2M.Q.LATCH x			Turn on or off ‘latching’ for MIDI notes received via MIDI in

OP	OP *(set)*	*(aliases)*	Description
I2M.Q.LC			Get the latest controller number (0..127) received via MIDI in
I2M.Q.LCC			Get the latest controller value (0..127) received via MIDI in
I2M.Q.LCH			Get the latest channel (1..16) received via MIDI in
I2M.Q.LN			Get the note number (0..127) of the latest Note On received via MIDI in
I2M.Q.LO			Get the note number (0..127) of the latest Note Off received via MIDI in
I2M.Q.LV			Get the velocity (1..127) of the latest Note On received via MIDI in
I2M.Q.NOTE x		**I2M.Q.N**	Get x (0..7) last note number (0..127) received via MIDI in
I2M.Q.VEL x		**I2M.Q.V**	Get x (0..7) last note velocity (1..127) received via MIDI in
I2M.RAT	**I2M.RAT x**		Get current ratcheting / Set ratcheting of MIDI notes to x ratchets (1..127) for current channel
I2M.RAT# ch x			Get current ratcheting / Set ratcheting of MIDI notes to x ratchets (1..127) for channel ch (0..32)
I2M.REP	**I2M.REP x**		Get current repetition / Set repetition of MIDI notes to x repetitions (1..127) for current channel
I2M.REP# ch x			Get current repetition / Set repetition of MIDI notes to x repetitions (1..127) for channel ch (0..32)
I2M.S# ch	**I2M.S# ch x**		Get current transposition / Set transposition of MIDI notes to x semitones (-127..127) for channel ch (0..32)
I2M.SHIFT	**I2M.SHIFT x**	**I2M.S**	Get current transposition / Set transposition of MIDI notes to x semitones (-127..127) for current channel
I2M.SOLO	**I2M.SOLO x**		Get solo state / Set solo state of current MIDI channel to x (0..1)

OP	OP *(set)*	*(aliases)*	Description
I2M.SOLO#	**I2M.SOLO# ch x**		Get solo state / Set solo state of MIDI channel ch to x (0..1)
I2M.START			Send MIDI Clock Start message
I2M.STOP			Send MIDI Clock Stop message
I2M.T# ch	**I2M.T# ch x**		Get current note duration / Set note duration of MIDI notes to x ms (0..32767) for channel ch (0..32).
I2M.TIME	**I2M.TIME x**	**I2M.T**	Get current note duration / Set note duration of MIDI notes to x ms (0..32767) for current channel
IF x: ...			if x is not zero execute command
IIA	**IIA address**		Set I2C address or get the currently selected address
IIB cmd			Execute the specified query and get a byte value back
IIB1 cmd value			Execute the specified query with 1 parameter and get a byte value back
IIB2 cmd value1 value2			Execute the specified query with 2 parameters and get a byte value back
IIB3 cmd value1 value2 value3			Execute the specified query with 3 parameters and get a byte value back
IIBB1 cmd value			Execute the specified query with 1 byte parameter and get a byte value back
IIBB2 cmd value1 value2			Execute the specified query with 2 byte parameters and get a byte value back
IIBB3 cmd value1 value2 value3			Execute the specified query with 3 byte parameters and get a byte value back
IIQ cmd			Execute the specified query and get a value back
IIQ1 cmd value			Execute the specified query with 1 parameter and get a value back
IIQ2 cmd value1 value2			Execute the specified query with 2 parameters and get a value back

OP	OP *(set)*	*(aliases)*	Description
IIQ3 cmd value1 value2 value3			Execute the specified query with 3 parameters and get a value back
IIQB1 cmd value			Execute the specified query with 1 byte parameter and get a value back
IIQB2 cmd value1 value2			Execute the specified query with 2 byte parameters and get a value back
IIQB3 cmd value1 value2 value3			Execute the specified query with 3 byte parameters and get a value back
IIS cmd			Execute the specified command
IIS1 cmd value			Execute the specified command with 1 parameter
IIS2 cmd value1 value2			Execute the specified command with 2 parameters
IIS3 cmd value1 value2 value3			Execute the specified command with 3 parameters
IISB1 cmd value			Execute the specified command with 1 byte parameter
IISB2 cmd value1 value2			Execute the specified command with 2 byte parameters
IISB3 cmd value1 value2 value3			Execute the specified command with 3 byte parameters
IN			Get the value of IN jack (0-16383)
IN.CAL.MAX			Reads the input CV and assigns the voltage to the max point
IN.CAL.MIN			Reads the input CV and assigns the voltage to the zero point
IN.CAL.RESET			Resets the input CV calibration
IN.SCALE min max			Set static scaling of the `IN` CV to between `min` and `max`.
INIT			clears all state data
INIT.CV x			clears all parameters on CV associated with output x
INIT.CV.ALL			clears all parameters on all CV's
INIT.DATA			clears all data held in all variables
INIT.P x			clears pattern number x
INIT.P.ALL			clears all patterns
INIT.SCENE			loads a blank scene

OP	OP *(set)*	*(aliases)*	Description
INIT.SCRIPT x			clear script number x
INIT.SCRIPT.ALL			clear all scripts
INIT.TIME x			clear time on trigger x
INIT.TR x			clear all parameters on trigger x
INIT.TR.ALL			clear all triggers
INR l x h		**><**	x is greater than l and less than h (within range)
INRI l x h		**>=<**	x is greater than or equal to l and less than or equal to h (within range, inclusive)
J	**J x**		get / set the variable J
JF.ADDR x			Sets JF II address (1 = primary, 2 = secondary). Use with only one JF on the bus! Saves to JF internal memory, so only one-time config is needed.
JF.CURVE			Gets value of CURVE knob.
JF.FM			Gets value of FM knob.
JF.GOD x			Redefines C3 to align with the 'God' note. x = 0 sets A to 440, x = 1 sets A to 432.
JF.INTONE			Gets value of INTONE knob and CV offset.
JF.MODE x			Set the current choice of standard functionality, or Just Type alternate modes (Speed switch to Sound for Synth, Shape for Geode). You'll likely want to put JF.MODE x in your Teletype INIT scripts. x = nonzero activates alternative modes. 0 restores normal.

OP	OP *(set)*	*(aliases)*	Description
JF.NOTE x y			Synth: polyphonically allocated note sequencing. Works as JF.VOX with chan selected automatically. Free voices will be taken first. If all voices are busy, will steal from the voice which has been active the longest. x = pitch relative to C3, y = velocity. Geode: works as JF.VOX with dynamic allocation of channel. Assigns the rhythmic stream to the oldest unused channel, or if all are busy, the longest running channel. x = division, y = number of repeats.
JF.PITCH x y			Change pitch without retriggering. x = channel, y = pitch relative to C3.
JF.POLY x y			As JF.NOTE but across dual JF. Switches between primary and secondary units every 6 notes or until reset using JF.POLY.RESET.
JF.POLY.RESET			Resets JF.POLY note count.
JF.QT x			When non-zero, all events are queued & delayed until the next quantize event occurs. Using values that don't align with the division of rhythmic streams will cause irregular patterns to unfold. Set to 0 to deactivate quantization. x = division, 0 deactivates quantization, 1 to 32 sets the subdivision & activates quantization.
JF.RAMP			Gets value of RAMP knob.
JF.RMODE x			Set the RUN state of Just Friends when no physical jack is present. (0 = run off, non-zero = run on)

OP	OP *(set)*	*(aliases)*	Description
JF.RUN x			Send a 'voltage' to the RUN input. Requires JF.RMODE 1 to have been executed, or a physical cable in JF's input. Thus Just Friend's RUN modes are accessible without needing a physical cable & control voltage to set the RUN parameter. use JF.RUN V x to set to x volts. The expected range is V -5 to V 5
JF.SEL x			Sets target JF unit (1 = primary, 2 = secondary).
JF.SHIFT x			Shifts the transposition of Just Friends, regardless of speed setting. Shifting by V 1 doubles the frequency in sound, or doubles the rate in shape. x = pitch, use N x for semitones, or V y for octaves.
JF.SPEED			Gets value of SPEED switch (1 = sound, 0 = shape).
JF.TICK x			Sets the underlying timebase of the Geode. x = clock. 0 resets the timebase to the start of measure. 1 to 48 shall be sent repetitively. The value representing ticks per measure. 49 to 255 sets beats-per-minute and resets the timebase to start of measure.
JF.TIME			Gets value of TIME knob and CV offset.
JF.TR x y			Simulate a TRIGGER input. x is channel (0 = all primary JF channels, 1..6 = primary JF, 7..12 = secondary JF, -1 = all channels both JF) and y is state (0 or 1)
JF.TSC			Gets value of MODE switch (0 = transient, 1 = sustain, 2 = cycle).
JF.TUNE x y z			Adjust the tuning ratios used by the INTONE control. x = channel, y = numerator (set the multiplier for the tuning ratio), z = denominator (set the divisor for the tuning ratio). JF.TUNE 0 0 0 resets to default ratios.

OP	OP *(set)*	*(aliases)*	Description
JF.VOX x y z			Synth mode: create a note at the specified channel, of the defined pitch & velocity. All channels can be set simultaneously with a chan value of 0. x = channel, y = pitch relative to C3, z = velocity (like JF.VTR). Geode mode: Create a stream of rhythmic envelopes on the named channel. x = channel, y = division, z = number of repeats.
JF.VTR x y			Like JF.TR with added volume control. Velocity is scaled with volts, so try V 5 for an output trigger of 5 volts. Channels remember their latest velocity setting and apply it regardless of TRIGGER origin (digital or physical). x = channel, 0 sets all channels. y = velocity, amplitude of output in volts. eg JF.VTR 1 V 4.
JF0: ...			Send following JF OPs to both units starting with selected unit.
JF1: ...			Send following JF OPs to unit 1 ignoring the currently selected unit.
JF2: ...			Send following JF OPs to unit 2 ignoring the currently selected unit.
JI x y			just intonation helper, precision ratio divider normalised to 1V
K	**K x**		get / set the variable K
KILL			clears stack, clears delays, cancels pulses, cancels slews, disables metronome
KR.CLK x			advance the clock for channel x (channel must have teletype clocking enabled)
KR.CUE	**KR.CUE x**		get/set the cued pattern
KR.CV x			get the current CV value for channel x
KR.DIR	**KR.DIR x**		get/set the step direction
KR.DUR x			get the current duration value for channel x

OP	OP *(set)*	*(aliases)*	Description
KR.L.LEN x y	**KR.L.LEN x y z**		get length of track x, parameter y / set to z
KR.L.ST x y	**KR.L.ST x y z**		get loop start for track x, parameter y / set to z
KR.MUTE x	**KR.MUTE x y**		get/set mute state for channel x (1 = muted, 0 = unmuted)
KR.PAT	**KR.PAT x**		get/set current pattern
KR.PERIOD	**KR.PERIOD x**		get/set internal clock period
KR.PG	**KR.PG x**		get/set the active page
KR.POS x y	**KR.POS x y z**		get/set position z for track z, parameter y
KR.PRE	**KR.PRE x**		return current preset / load preset x
KR.RES x y			reset position to loop start for track x, parameter y
KR.SCALE	**KR.SCALE x**		get/set current scale
KR.TMUTE x			toggle mute state for channel x
L x y: ...			run the command sequentially with I values from x to y
LAST x			get value in milliseconds since last script run time
LIM x y z			limit the value x to the range y to z inclusive
LIVE.DASH x		**LIVE.D**	Show the dashboard with index x
LIVE.GRID		**LIVE.G**	Show grid visualizer in live mode
LIVE.OFF		**LIVE.O**	Show the default live mode screen
LIVE.VARS		**LIVE.V**	Show variables in live mode
LROT x y		**<<<**	circular left shift x by y bits, wrapping around when bits fall off the end
LSH x y		**<<**	left shift x by y bits, in effect multiply x by 2 to the power of y
LT x y		**<**	x is less than y
LTE x y		**<=**	x is less than or equal to y
LV.CV x			get the current CV value for channel x
LV.L.DIR	**LV.L.DIR x**		get/set loop direction
LV.L.LEN	**LV.L.LEN x**		get/set loop length
LV.L.ST	**LV.L.ST x**		get/set loop start
LV.POS	**LV.POS x**		get/set current position

OP	OP *(set)*	*(aliases)*	Description
LV.PRE	**LV.PRE x**		return current preset / load preset x
LV.RES x			reset, 0 for soft reset (on next ext. clock), 1 for hard reset
M	**M x**		get/set metronome interval to x (in ms), default 1000, minimum value 25
M!	**M! x**		get/set metronome to experimental interval x (in ms), minimum value 2
M.ACT	**M.ACT x**		get/set metronome activation to x (0/1), default 1 (enabled)
M.RESET			hard reset metronome count without triggering
MA.CLR			clear all connections
MA.COL col	**MA.COL col value**		get or set column col (as a 16 bit unsigned value where each bit represents a connection)
MA.OFF x y			disconnect row x and column y in the current program
MA.ON x y			connect row x and column y in the current program (rows/columns are 0-based)
MA.PCLR pgm			clear all connections in program pgm
MA.PCOL pgm col	**MA.PCOL pgm col value**		get or set column col in program pgm
MA.PGM pgm			select the current program (1-based)
MA.POFF x y pgm			connect row x and column y in program pgm
MA.PON pgm x y			connect row x and column y in program pgm
MA.PROW pgm row	**MA.PROW pgm row value**		get or set row row in program pgm
MA.PSET pgm x y state			set the connection at row x and column y in program pgm to state (1 - on, 0 - off)
MA.RESET			reset program sequencer
MA.ROW row	**MA.ROW row value**		get or set row row
MA.SELECT x			select the default matrixarchate module, default 1

OP	OP *(set)*	*(aliases)*	Description
MA.SET x y state			set the connection at row x and column y to `state` (1 - on, 0 - off)
MA.STEP			advance program sequencer
MAX x y			return the maximum of x and y
ME.CV x			get the current CV value for channel x
ME.PERIOD	**ME.PERIOD x**		get/set internal clock period
ME.PRE	**ME.PRE x**		return current preset / load preset x
ME.RES x			reset channel x (`0` = all), also used as “start”
ME.SCALE	**ME.SCALE x**		get/set current scale
ME.STOP x			stop channel x (`0` = all)
MI.$ x	**MI.$ x y**		assign MIDI event type x to script y
MI.C			get the controller number (0..127) at index specified by variable `I`
MI.CC			get the controller value (0..127) at index specified by variable `I`
MI.CCH			get the controller event channel (1..16) at index specified by variable `I`
MI.CCV			get the controller value scaled to 0..+10V range at index specified by variable `I`
MI.CL			get the number of controller events
MI.CLKD	**MI.CLKD x**		set clock divider to x (1-24) or get the current divider
MI.CLKR			reset clock counter
MI.LC			get the latest controller number (0..127)
MI.LCC			get the latest controller value (0..127)
MI.LCCV			get the latest controller value scaled to 0..16383 range (0..+10V)
MI.LCH			get the latest channel (1..16)
MI.LE			get the latest event type
MI.LN			get the latest Note On (0..127)

OP	OP *(set)*	*(aliases)*	Description
MI.LNV			get the latest Note On scaled to teletype range (shortcut for `N MI.LN`)
MI.LO			get the latest Note Off (0..127)
MI.LV			get the latest velocity (0..127)
MI.LVV			get the latest velocity scaled to 0..16383 range (0..+10V)
MI.N			get the Note On (0..127) at index specified by variable `I`
MI.NCH			get the Note On event channel (1..16) at index specified by variable `I`
MI.NL			get the number of Note On events
MI.NV			get the Note On scaled to 0..+10V range at index specified by variable `I`
MI.O			get the Note Off (0..127) at index specified by variable `I`
MI.OCH			get the Note Off event channel (1..16) at index specified by variable `I`
MI.OL			get the number of Note Off events
MI.V			get the velocity (0..127) at index specified by variable `I`
MI.VV			get the velocity scaled to 0..10V range at index specified by variable `I`
MID.SHIFT o			shift pitch CV by standard Teletype pitch value (e.g. `N 6`, `V -1`, etc)
MID.SLEW t			set pitch slew time in ms (applies to all allocation styles except FIXED)
MIN x y			return the minimum of x and y
MOD x y		**%**	find the remainder after division of x by y
MP.PRESET x			set Meadowphysics to preset x (indexed from `0`)
MP.RESET x			reset countdown for channel x (`0` = all, `1`–`8` = individual channels)
MP.STOP x			reset channel x (`0` = all, `1`–`8` = individual channels)

OP	OP *(set)*	*(aliases)*	Description
MUL x y		*	multiply x and y together
MUTE x	**MUTE x y**		Disable trigger input x
N x			converts an equal temperament note number to a value usable by the CV outputs (x in the range -127 to 127)
N.B d	**N.B r s**		get degree d of scale/set scale root to r, scale to s, s is either bit mask (s >= 1) or scale preset (s < 1)
N.BX i d	**N.BX i r s**		multi-index version of N.B, scale at i (index) 0 is shared with N.B
N.C r c d			Note Chord operator, r is the root note (0-127), c is the chord (0-12) and d is the degree (0-3), returns a value from the N table.
N.CS r s d c			Note Chord Scale operator, r is the root note (0-127), s is the scale (0-8), d is the scale degree (1-7) and c is the chord component (0-3), returns a value from the N table.
N.S r s d			Note Scale operator, r is the root note (0-127), s is the scale (0-8) and d is the degree (1-7), returns a value from the N table.
NE x y		**!=** , **XOR**	x is not equal to y
NR p m f s			Numeric Repeater, p is prime pattern (0-31), m is & mask (0-3), f is variation factor (0-16) and s is step (0-15), returns 0 or 1
NZ x			x is not 0
O	**O x**		auto-increments *after* each access, can be set, starting value 0
O.INC	**O.INC x**		how much to increment O by on each invocation, default 1
O.MAX	**O.MAX x**		the upper bound for O, default 63
O.MIN	**O.MIN x**		the lower bound for O, default 0
O.WRAP	**O.WRAP x**		should O wrap when it reaches its bounds, default 1
OR x y		||	logical OR of x and y
OR.BANK x			Select preset bank x (1-8)

OP	OP *(set)*	*(aliases)*	Description
OR.CLK x			Advance track x (1-4)
OR.CVA x			Select tracks for CV A where x is a binary number representing the tracks
OR.CVB x			Select tracks for CV B where x is a binary number representing the tracks
OR.DIV x			Set divisor for selected track to x (1-16)
OR.GRST x			Global reset (x can be any value)
OR.MUTE x			Mute trigger selected by OR.TRK (0 = on, 1 = mute)
OR.PHASE x			Set phase for selected track to x (0-16)
OR.PRESET x			Select preset x (1-8)
OR.RELOAD x			Reload preset or bank (0 - current preset, 1 - current bank, 2 - all banks)
OR.ROTS x			Rotate scales by x (1-15)
OR.ROTW x			Rotate weights by x (1-3)
OR.RST x			Reset track x (1-4)
OR.SCALE x			Select scale x (1-16)
OR.TRK x			Choose track x (1-4) to be used by OR.DIV, OR.PHASE, OR.WGT or OR.MUTE
OR.WGT x			Set weight for selected track to x (1-8)
OR3 x y z		|||	logical OR of x, y and z
OR4 x y z a		||||	logical OR of x, y, z and a
OTHER: ...			runs the command when the previous EVERY/SKIP did not run its command.
OUTR l x h		**<>**	x is less than l or greater than h (out of range)
OUTRI l x h		**<=>**	x is less than or equal to l or greater than or equal to h (out of range, inclusive)
P x	**P x y**		get/set the value of the working pattern at index x
P.+ x y			increase the value of the working pattern at index x by y
P.+W x y a b			increase the value of the working pattern at index x by y and wrap it to a..b range

OP	OP *(set)*	*(aliases)*	Description
P.- x y			decrease the value of the working pattern at index x by y
P.-W x y a b			decrease the value of the working pattern at index x by y and wrap it to a..b range
P.END	**P.END x**		get/set the end location of the working pattern, default 63
P.HERE	**P.HERE x**		get/set value at current index of working pattern
P.I	**P.I x**		get/set index position for the working pattern.
P.INS x y			insert value y at index x of working pattern, shift later values down, destructive to loop length
P.L	**P.L x**		get/set pattern length of the working pattern, non-destructive to data
P.MAP: ...			apply the 'function' to each value in the active pattern, `I` takes each pattern value
P.MAX			find the first maximum value in the pattern between the START and END for the working pattern and return its index
P.MIN			find the first minimum value in the pattern between the START and END for the working pattern and return its index
P.N	**P.N x**		get/set the pattern number for the working pattern, default `0`
P.NEXT	**P.NEXT x**		increment index of working pattern then get/set value
P.POP			return and remove the value from the end of the working pattern (like a stack), destructive to loop length
P.PREV	**P.PREV x**		decrement index of working pattern then get/set value
P.PUSH x			insert value x to the end of the working pattern (like a stack), destructive to loop length
P.REV			reverse the values in the active pattern (between its START and END)

OP	OP *(set)*	*(aliases)*	Description
P.RM x			delete index x of working pattern, shift later values up, destructive to loop length
P.RND			return a value randomly selected between the start and the end position
P.ROT n			rotate the values in the active pattern n steps (between its START and END, negative rotates backward)
P.SEED	**P.SEED x**	**P.SD**	get / set the random number generator seed for the `P.RND` and `PN.RND` ops
P.SHUF			shuffle the values in active pattern (between its START and END)
P.START	**P.START x**		get/set the start location of the working pattern, default `0`
P.WRAP	**P.WRAP x**		when the working pattern reaches its bounds does it wrap (`0`/1), default 1 (enabled)
PARAM		**PRM**	Get the value of PARAM knob (0-16383)
PARAM.CAL.MAX			Reads the Parameter Knob maximum position and assigns the maximum point
PARAM.CAL.MIN			Reads the Parameter Knob minimum position and assigns a zero value
PARAM.CAL.RESET			Resets the Parameter Knob calibration
PARAM.SCALE min max			Set static scaling of the PARAM knob to between `min` and `max`.
PN x y	**PN x y z**		get/set the value of pattern x at index y
PN.+ x y z			increase the value of pattern x at index y by z
PN.+W x y z a b			increase the value of pattern x at index y by z and wrap it to a..b range
PN.- x y z			decrease the value of pattern x at index y by z
PN.-W x y z a b			decrease the value of pattern x at index y by z and wrap it to a..b range

OP	OP *(set)*	*(aliases)*	Description
PN.END x	**PN.END x y**		get/set the end location of the pattern x, default 63
PN.HERE x	**PN.HERE x y**		get/set value at current index of pattern x
PN.I x	**PN.I x y**		get/set index position for pattern x
PN.INS x y z			insert value z at index y of pattern x, shift later values down, destructive to loop length
PN.L x	**PN.L x y**		get/set pattern length of pattern x. non-destructive to data
PN.MAP x: ...			apply the 'function' to each value in pattern x, I takes each pattern value
PN.MAX x			find the first maximum value in the pattern between the START and END for pattern x and return its index
PN.MIN x			find the first minimum value in the pattern between the START and END for pattern x and return its index
PN.NEXT x	**PN.NEXT x y**		increment index of pattern x then get/set value
PN.POP x			return and remove the value from the end of pattern x (like a stack), destructive to loop length
PN.PREV x	**PN.PREV x y**		decrement index of pattern x then get/set value
PN.PUSH x y			insert value y to the end of pattern x (like a stack), destructive to loop length
PN.REV x			reverse the values in pattern x
PN.RM x y			delete index y of pattern x, shift later values up, destructive to loop length
PN.RND x			return a value randomly selected between the start and the end position of pattern x
PN.ROT x n			rotate the values in pattern x (between its START and END, negative rotates backward)
PN.SHUF x			shuffle the values in pattern x (between its START and END)

OP	OP *(set)*	*(aliases)*	Description
PN.START x	**PN.START x y**		get/set the start location of pattern x, default 0
PN.WRAP x	**PN.WRAP x y**		when pattern x reaches its bounds does it wrap (0/1), default 1 (enabled)
PRINT x	**PRINT x y**	**PRT**	Print a value on a live mode dashboard or get the printed value
PROB x: ...			potentially execute command with probability x (0-100)
PROB.SEED	**PROB.SEED x**	**PROB.SD**	get / set the random number generator seed for the PR0B mod
Q	**Q x**		Modify the queue entries
Q.2P	**Q.2P i**		Copy queue to current pattern/copy queue to pattern at index i
Q.ADD x	**Q.ADD x i**		Perform addition on elements in queue
Q.AVG	**Q.AVG x**		Return the average of the queue
Q.CLR	**Q.CLR x**		Clear queue
Q.DIV x	**Q.DIV x i**		Perform division on elements in queue
Q.GRW	**Q.GRW x**		Get/set grow state
Q.I i	**Q.I i x**		Get/set value of elements at index
Q.MAX	**Q.MAX x**		Get/set maximum value
Q.MIN	**Q.MIN x**		Get/set minimum value
Q.MOD x	**Q.MOD x i**		Perform module (%) on elements in queue
Q.MUL x	**Q.MUL x i**		Perform multiplication on elements in queue
Q.N	**Q.N x**		The queue length
Q.P2	**Q.P2 i**		Copy current pattern to queue/copy pattern at index i to queue
Q.REV			Reverse queue
Q.RND	**Q.RND x**		Get random element/randomize elements
Q.SH	**Q.SH x**		Shift elements in queue
Q.SRT	**Q.SRT**		Sort all or part of queue
Q.SUB x	**Q.SUB x i**		Perform subtraction on elements in queue

OP	OP *(set)*	*(aliases)*	Description
Q.SUM	**Q.SUM x**		Get sum of elements
QT x y			round x to the closest multiple of y (quantise)
QT.B x			quantize 1V/OCT signal x to scale defined by N.B
QT.BX i x			quantize 1V/OCT signal x to scale defined by N.BX in scale index i
QT.CS x r s d c			quantize 1V/OCT signal x to chord c (1-7) from scale s (0-8, reference N.S scales) at degree d (1-7) with root 1V/OCT pitch r
QT.S x r s			quantize 1V/OCT signal x to scale s (0-8, reference N.S scales) with root 1V/OCT pitch r
R	**R x**		get a random number/set R.MIN and R.MAX to same value x (effectively allowing R to be used as a global variable)
R.MAX x			set the upper end of the range from -32768 – 32767, default: 16383
R.MIN x			set the lower end of the range from -32768 – 32767, default: 0
RAND x		**RND**	generate a random number between 0 and x inclusive
RAND.SEED	**RAND.SEED x**	**RAND.SD**, **R.SD**	get / set the random number generator seed for R, RRAND, and RAND ops
RRAND x y		**RRND**	generate a random number between x and y inclusive
RROT x y		**>>>**	circular right shift x by y bits, wrapping around when bits fall off the end
RSH x y		**>>**	right shift x by y bits, in effect divide x by 2 to the power of y
S: ...			Place a command onto the stack
S.ALL			Execute all entries in the stack
S.CLR			Clear all entries in the stack
S.L			Get the length of the stack
S.POP			Execute the most recent entry
SC.CV x y			CV target value for the ER-301 virtual output x to value y

OP	OP *(set)*	*(aliases)*	Description
SC.CV.OFF x y			CV offset added to the ER-301 virtual output x
SC.CV.SET x			Set CV value for the ER-301 virtual output x
SC.CV.SLEW x y			Set the CV slew time for the ER-301 virtual output x in ms
SC.TR x y			Set trigger output for the ER-301 virtual output x to y (0-1)
SC.TR.POL x y			Set polarity of trigger for the ER-301 virtual output x to y (0-1)
SC.TR.PULSE x		**SC.TR.P**	Pulse the ER-301 virtual trigger output x
SC.TR.TIME x y			Set the pulse time for the ER-301 virtual trigger x to y in ms
SC.TR.TOG x			Flip the state for the ER-301 virtual trigger output x
SCALE a b x y i		**SCL**	scale `i` from range a to b to range x to y, i.e. `i * (y - x) / (b - a)`
SCALE a b i		**SCL0**	scale `i` from range `0` to a to range `0` to b
SCENE	**SCENE x**		get the current scene number, or load scene x (0-31)
SCENE.G x			load scene x (0-31) without loading grid control states
SCENE.P x			load scene x (0-31) without loading pattern state
SCRIPT	**SCRIPT x**	**$**	get current script number, or execute script x (1-10), recursion allowed
SCRIPT.POL x	**SCRIPT.POL x p**	**$.POL**	get script x trigger polarity, or set polarity p (1 rising edge, 2 falling, 3 both)
SEED	**SEED x**		get / set the random number generator seed for all SEED ops
SGN x			sign function: 1 for positive, -1 for negative, 0 for 0
SKIP x: ...			run the command every time except the xth time.
STATE x			Read the current state of input x
SUB x y		**-**	subtract y from x
SYNC x			synchronizes *all* EVERY and SKIP counters to offset x.

OP	OP *(set)*	*(aliases)*	Description
T	**T x**		get / set the variable T, typically used for time, default 0
TI.IN x			reads the value of IN jack x; default return range is from -16384 to 16383 - representing -10V to +10V; return range can be altered by the TI.IN.MAP command
TI.IN.CALIB x y			calibrates the scaling for IN jack x; y of -1 sets the -10V point; y of 0 sets the 0V point; y of 1 sets the +10V point
TI.IN.INIT x			initializes IN jack x back to the default boot settings and behaviors; neutralizes mapping (but not calibration)
TI.IN.MAP x y z			maps the IN values for input jack x across the range y - z (default range is -16384 to 16383 - representing -10V to +10V)
TI.IN.N x			return the quantized note number for IN jack x using the scale set by TI.IN.SCALE
TI.IN.QT x			return the quantized value for IN jack x using the scale set by TI.IN.SCALE; default return range is from -16384 to 16383 - representing -10V to +10V
TI.IN.SCALE x			select scale # y for IN jack x; scales listed in full description
TI.INIT d			initializes all of the PARAM and IN inputs for device number d (1-8)
TI.PARAM x		**TI.PRM**	reads the value of PARAM knob x; default return range is from 0 to 16383; return range can be altered by the TI.PARAM.MAP command
TI.PARAM.CALIB x y		**TI.PRM.CALIB**	calibrates the scaling for PARAM knob x; y of 0 sets the bottom bound; y of 1 sets the top bound
TI.PARAM.INIT x		**TI.PRM.INIT**	initializes PARAM knob x back to the default boot settings and behaviors; neutralizes mapping (but not calibration)

OP	OP *(set)*	*(aliases)*	Description
TI.PARAM.MAP x y z		**TI.PRM.MAP**	maps the PARAM values for input x across the range y - z (defaults 0-16383)
TI.PARAM.N x		**TI.PRM.N**	return the quantized note number for PARAM knob x using the scale set by TI.PARAM.SCALE
TI.PARAM.QT x		**TI.PRM.QT**	return the quantized value for PARAM knob x using the scale set by TI.PARAM.SCALE; default return range is from 0 to 16383
TI.PARAM.SCALE x		**TI.PRM.SCALE**	select scale # y for PARAM knob x; scales listed in full description
TI.RESET d			resets the calibration data for TXi number d (1-8) to its factory defaults (no calibration)
TI.STORE d			stores the calibration data for TXi number d (1-8) to its internal flash memory
TIME	**TIME x**		timer value, counts up in ms., wraps after 32s, can be set
TIME.ACT	**TIME.ACT x**		enable or disable timer counting, default 1
TO.CV x			CV target output x; y values are bipolar (-16384 to +16383) and map to -10 to +10
TO.CV.CALIB x			Locks the current offset (CV.OFF) as a calibration offset and saves it to persist between power cycles for output x.
TO.CV.INIT x			initializes CV output x back to the default boot settings and behaviors; neutralizes offsets, slews, envelopes, oscillation, etc.
TO.CV.LOG x y			translates the output for CV output x to logarithmic mode y; y defaults to 0 (off); mode 1 is for 0-16384 (0V-10V), mode 2 is for 0-8192 (0V-5V), mode 3 is for 0-4096 (0V-2.5V), etc.
TO.CV.N x y			target the CV to note y for output x; y is indexed in the output's current CV.SCALE

OP	OP *(set)*	*(aliases)*	Description
TO.CV.N.SET x y			set the CV to note y for output x; y is indexed in the output's current CV.SCALE (ignoring SLEW)
TO.CV.OFF x y			set the CV offset for output x; y values are added at the final stage
TO.CV.QT x y			CV target output x; y is quantized to output's current CV.SCALE
TO.CV.QT.SET x y			set the CV for output x (ignoring SLEW); y is quantized to output's current CV.SCALE
TO.CV.RESET x			Clears the calibration offset for output x
TO.CV.SCALE x y			select scale # y for CV output x; scales listed in full description
TO.CV.SET x y			set the CV for output x (ignoring SLEW); y values are bipolar (-16384 to +16383) and map to -10 to +10
TO.CV.SLEW x y			set the slew amount for output x; y in milliseconds
TO.CV.SLEW.M x y			set the slew amount for output x; y in minutes
TO.CV.SLEW.S x y			set the slew amount for output x; y in seconds
TO.ENV x y			trigger the attack stage of output x when y changes to 1, or decay stage when it changes to 0
TO.ENV.ACT x y			activates/deactivates the AD envelope generator for the CV output x; y turns the envelope generator off (0 - default) or on (1); CV amplitude is used as the peak for the envelope and needs to be > 0 for the envelope to be perceivable
TO.ENV.ATT x y			set the envelope attack time to y for CV output x; y in milliseconds (default 12 ms)
TO.ENV.ATT.M x y			set the envelope attack time to y for CV output x; y in minutes
TO.ENV.ATT.S x y			set the envelope attack time to y for CV output x; y in seconds

OP	OP *(set)*	*(aliases)*	Description
TO.ENV.DEC x y			set the envelope decay time to y for CV output x; y in milliseconds (default 250 ms)
TO.ENV.DEC.M x y			set the envelope decay time to y for CV output x; y in minutes
TO.ENV.DEC.S x y			set the envelope decay time to y for CV output x; y in seconds
TO.ENV.EOC x n			at the end of cycle of CV output x, fires a PULSE to the trigget output n
TO.ENV.EOR x n			at the end of rise of CV output x, fires a PULSE to the trigget output n
TO.ENV.LOOP x y			causes the envelope on CV output x to loop for y times
TO.ENV.TRIG x			triggers the envelope at CV output x to cycle; CV amplitude is used as the peak for the envelope and needs to be > 0 for the envelope to be perceivable
TO.INIT d			initializes all of the TR and CV outputs for device number d (1-8)
TO.KILL d			cancels all TR pulses and CV slews for device number d (1-8)
TO.M d y			sets the 4 independent metronome intervals for device d (1-8) to y in milliseconds; default 1000
TO.M.ACT d y			sets the active status for the 4 independent metronomes on device d (1-8) to y (0/1); default 0 (disabled)
TO.M.BPM d y			sets the 4 independent metronome intervals for device d to y in Beats Per Minute
TO.M.COUNT d y			sets the number of repeats before deactivating for the 4 metronomes on device d to y; default 0 (infinity)
TO.M.M d y			sets the 4 independent metronome intervals for device d to y in minutes

OP	OP *(set)*	*(aliases)*	Description
TO.M.S d y			sets the 4 independent metronome intervals for device d to y in seconds; default 1
TO.M.SYNC d			synchronizes the 4 metronomes for device number d (1-8)
TO.OSC x y			Targets oscillation for CV output x to y
TO.OSC.CTR x y			centers the oscillation on CV output x to y; y values are bipolar (-16384 to +16383) and map to -10 to +10
TO.OSC.CYC x y			targets the oscillator cycle length to y for CV output x with the portamento rate determined by the TO.OSC.SLEW value; y is in milliseconds
TO.OSC.CYC.M x y			targets the oscillator cycle length to y for CV output x with the portamento rate determined by the TO.OSC.SLEW value; y is in minutes
TO.OSC.CYC.M.SET x y			sets the oscillator cycle length to y for CV output x (ignores CV.OSC.SLEW); y is in minutes
TO.OSC.CYC.S x y			targets the oscillator cycle length to y for CV output x with the portamento rate determined by the TO.OSC.SLEW value; y is in seconds
TO.OSC.CYC.S.SET x y			sets the oscillator cycle length to y for CV output x (ignores CV.OSC.SLEW); y is in seconds
TO.OSC.CYC.SET x y			sets the oscillator cycle length to y for CV output x (ignores CV.OSC.SLEW); y is in milliseconds
TO.OSC.FQ x y			targets oscillation for CV output x to frequency y in Hertz
TO.OSC.FQ.SET x y			targets oscillation for CV output x to frequency y in Hertz (ignores slew)
TO.OSC.LFO x y			Targets oscillation for CV output x to LFO frequency y in millihertz

OP	OP *(set)*	*(aliases)*	Description
TO.OSC.LFO.SET x y			Targets oscillation for CV output x to LFO frequency y in millihertz (ignores slew)
TO.OSC.N x y			targets oscillation for CV output x to note y
TO.OSC.N.SET x y			sets oscillation for CV output x to note y (ignores slew)
TO.OSC.PHASE x y			sets the phase offset of the oscillator on CV output x to y (0 to 16383); y is the range of one cycle
TO.OSC.QT x y			targets oscillation for CV output x to y
TO.OSC.QT.SET x y			set oscillation for CV output x to y, quantized to the current scale (ignores slew)
TO.OSC.RECT x y			rectifies the polarity of the oscillator for output x to y; 0 is no rectification, +/-1 is partial rectification, +/-2 is full rectification
TO.OSC.SCALE x y			select scale # y for CV output x; scales listed in full description
TO.OSC.SET x y			set oscillation for CV output x to y (ignores slew)
TO.OSC.SLEW x y			sets the frequency slew time (portamento) for the oscillator on CV output x to y; y in milliseconds
TO.OSC.SLEW.M x y			sets the frequency slew time (portamento) for the oscillator on CV output x to y; y in minutes
TO.OSC.SLEW.S x y			sets the frequency slew time (portamento) for the oscillator on CV output x to y; y in seconds
TO.OSC.SYNC x			resets the phase of the oscillator on CV output x (relative to `TO.OSC.PHASE`)
TO.OSC.WAVE x y			set the waveform for output x to y; y range is `0-4500`, blending between 45 waveforms

OP	OP *(set)*	*(aliases)*	Description
TO.OSC.WIDTH x y			sets the width of the pulse wave on output x to y; y is a percentage of total width (0 to 100); only affects waveform `3000`
TO.TR x y			sets the TR value for output x to y (0/1)
TO.TR.INIT x			initializes TR output x back to the default boot settings and behaviors; neutralizes metronomes, dividers, pulse counters, etc.
TO.TR.M x y			sets the independent metronome interval for output x to y in milliseconds; default `1000`
TO.TR.M.ACT x y			sets the active status for the independent metronome for output x to y (`0`/1); default `0` (disabled)
TO.TR.M.BPM x y			sets the independent metronome interval for output x to y in Beats Per Minute
TO.TR.M.COUNT x y			sets the number of repeats before deactivating for output x to y; default `0` (infinity)
TO.TR.M.M x y			sets the independent metronome interval for output x to y in minutes
TO.TR.M.MUL x y			multiplies the `M` rate on TR output x by y; y defaults to 1 - no multiplication
TO.TR.M.S x y			sets the independent metronome interval for output x to y in seconds; default 1
TO.TR.M.SYNC x			synchronizes the PULSE for metronome on TR output number x
TO.TR.POL x y			sets the polarity for TR output n
TO.TR.PULSE x		**TO.TR.P**	pulses the TR value for output x for the duration set by `TO.TR.TIME/S/M`
TO.TR.PULSE.DIV x y		**TO.TR.P.DIV**	sets the clock division factor for TR output x to y
TO.TR.PULSE.MUTE x y		**TO.TR.P.MUTE**	mutes or un-mutes TR output x; y is 1 (mute) or 0 (un-mute)

OP	OP *(set)*	*(aliases)*	Description
TO.TR.TIME x y			sets the time for TR.PULSE on output n; y in milliseconds
TO.TR.TIME.M x y			sets the time for TR.PULSE on output n; y in minutes
TO.TR.TIME.S x y			sets the time for TR.PULSE on output n; y in seconds
TO.TR.TOG x			toggles the TR value for output x
TO.TR.WIDTH x y			sets the time for TR.PULSE on output n based on the width of its current metronomic value; y in percentage (0-100)
TOSS			randomly return 0 or 1
TOSS.SEED	**TOSS.SEED x**	**TOSS.SD**	get / set the random number generator seed for the TOSS op
TR x	**TR x y**		Set trigger output x to y (0-1)
TR.POL x	**TR.POL x y**		Set polarity of trigger output x to y (0-1)
TR.PULSE x		**TR.P**	Pulse trigger output x
TR.TIME x	**TR.TIME x y**		Set the pulse time of trigger x to y ms
TR.TOG x			Flip the state of trigger output x
V x			converts a voltage to a value usable by the CV outputs (x between 0 and 10)
VN x			converts 1V/OCT value x to an equal temperament note number
VV x			converts a voltage to a value usable by the CV outputs (x between 0 and 1000, 100 represents 1V)
W x: ...			run the command while condition x is true
W/.SEL x			Sets target W/2.0 unit (1 = primary, 2 = secondary).
W/1: ...			Send following W/2.0 OPs to unit 1 ignoring the currently selected unit.
W/2: ...			Send following W/2.0 OPs to unit 2 ignoring the currently selected unit.
W/D.CLK			receive clock pulse for synchronization
W/D.CLK.RATIO mul div			set clock pulses per buffer time, with clock mul/div (s8)

OP	OP *(set)*	*(aliases)*	Description
W/D.CUT count			jump to loop location as a fraction of loop length (u8)
divisions			
W/D.FBK level			amount of feedback from read head to write head (s16V)
W/D.FILT cutoff			centre frequency of filter in feedback loop (s16V)
W/D.FREEZE			deactivate record head to freeze the current buffer (s8)
is_active			
W/D.FREQ volts			manipulate tape speed with musical values (s16V)
W/D.FREQ.RNG			TBD (s8)
freq_range			
W/D.LEN count			set buffer loop length as a fraction of buffer time (u8)
divisions			
W/D.MIX fade			fade from dry to delayed signal
W/D.MOD.AMT			set the `amount` (s16V) of delay line modulation to be applied
amount			
W/D.MOD.RATE			set the multiplier for the modulation rate (s16V)
rate			
W/D.PLUCK			pluck the delay line with noise at volume (s16V)
volume			
W/D.POS count			set loop start location as a fraction of buffer time (u8)
divisions			
W/D.RATE			direct `multiplier` (s16V) of tape speed
multiplier			
W/D.TIME			set delay buffer length in `seconds` (s16V), when rate == 1
seconds			
W/S.AR.MODE			in attack-release mode, all notes are `plucked` and no `release` is required’
is_ar			
W/S.CURVE curve			cross-fade waveforms: -5=square, 0=triangle, 5=sine (s16V)
W/S.FM.ENV			amount of vactrol envelope applied to fm index, -5 to +5 (s16V)
amount			
W/S.FM.INDEX			amount of FM modulation. -5=negative, 0=minimum, 5=maximum (s16V)
index			
W/S.FM.RATIO			ratio of the FM modulator to carrier as a ratio. floating point values up to 20.0 supported (s16V)
num den			

OP	OP *(set)*	*(aliases)*	Description
W/S.LPG.SYM symmetry			vactrol attack-release ratio. -5=fastest attack, 5=long swells (s16V)
W/S.LPG.TIME time			vactrol `time` (s16V) constant. -5=drones, 0=vtl5c3, 5=blits
W/S.NOTE pitch level			dynamically assign a voice, set to `pitch` (s16V), strike with `velocity`(s16V)
W/S.PATCH jack param			patch a hardware `jack` (s8) to a `param` (s8) destination
W/S.PITCH voice pitch			set `voice` (s8) to `pitch` (s16V) in volts-per-octave
W/S.POLY pitch level			As `W/S.NOTE` but across dual W/. Switches between primary and secondary units every 4 notes or until reset using `W/S.POLY.RESET`.
W/S.POLY.RESET			Resets `W/S.POLY` note count.
W/S.RAMP ramp			waveform symmetry: -5=rampwave, 0=triangle, 5=sawtooth (NB: affects FM tone)
W/S.VEL voice velocity			strike the vactrol of `voice` (s8) at `velocity` (s16V) in volts
W/S.VOICES count			set number of polyphonic voices to allocate. use 0 for unison mode (s8)
W/S.VOX voice pitch velocity			set `voice` (s8) to `pitch` (s16V) and strike the vactrol at `velocity` (s16V)
W/T.CLEARTAPE			WARNING! Erases all recorded audio on the tape!
W/T.ECHOMODE is_echo			Set to 1 to playback before erase. 0 (default) erases first (s8)
W/T.ERASE.LVL level			Strength of erase head when recording. 0 is overdub, 1 is overwrite. Opposite of feedback (s16V)
W/T.FREQ freq			Set speed as a `frequency` (s16V) style value. Maintains reverse state
W/T.LOOP.ACTIVE state			Set the state of looping (s8)
W/T.LOOP.END			Set the current time as the loop end, and jump to start

OP	OP *(set)*	*(aliases)*	Description
W/T.LOOP.NEXT direction			Move loop brace forward/backward by length of loop. Zero jumps to loop start (s8)
W/T.LOOP.SCALE scale			Mul(Positive) or Div(Negative) loop brace by arg. Zero resets to original window (s8)
W/T.LOOP.START			Set the current time as the beginning of a loop
W/T.MONITOR.LVL gain			Level of input passed directly to output (s16V)
W/T.PLAY playback			Set the playback state. -1 will flip playback direction (s8)
W/T.REC active			Sets recording state to `active` (s8)
W/T.REC.LVL gain			Level of input material recorded to tape (s16V)
W/T.REV			Reverse the direction of playback
W/T.SEEK seconds sub			Move playhead relative to current position (s16)
W/T.SPEED speed deno			Set speed as a rate, or ratio. Negative values are reverse (s16V)
W/T.TIME seconds sub			Move playhead to an arbitrary location on tape (s16)
WRAP x y z		**WRP**	limit the value x to the range y to z inclusive, but with wrapping
WS.CUE x			Go to a cuepoint relative to the playhead position. `0` retriggers the current location. 1 jumps to the next cue forward. `-1` jumps to the previous cue in the reverse. These actions are relative to playback direction such that `0` always retriggers the most recently passed location
WS.LOOP x			Set the loop state on/off. `0` is off. Any other value turns loop on
WS.PLAY x			Set playback state and direction. `0` stops playback. 1 sets forward motion, while `-1` plays in reverse

OP	OP *(set)*	*(aliases)*	Description
WS.REC x			Set recording mode. 0 is playback only. 1 sets overdub mode for additive recording. -1 sets overwrite mode to replace the tape with your input
WW.END x			Set the loop end position (0-15)
WW.MUTE1 x			Mute trigger 1 (0 = on, 1 = mute)
WW.MUTE2 x			Mute trigger 2 (0 = on, 1 = mute)
WW.MUTE3 x			Mute trigger 3 (0 = on, 1 = mute)
WW.MUTE4 x			Mute trigger 4 (0 = on, 1 = mute)
WW.MUTEA x			Mute CV A (0 = on, 1 = mute)
WW.MUTEB x			Mute CV B (0 = on, 1 = mute)
WW.PATTERN x			Change pattern (0-15)
WW.PMODE x			Set the loop play mode (0-5)
WW.POS x			Cut to position (0-15)
WW.PRESET x			Recall preset (0-7)
WW.QPATTERN x			Change pattern (0-15) after current pattern ends
WW.START x			Set the loop start position (0-15)
WW.SYNC x			Cut to position (0-15) and hard-sync the clock (if clocked internally)
X	**X x**		get / set the variable X, default 0
Y	**Y x**		get / set the variable Y, default 0
Z	**Z x**		get / set the variable Z, default 0
^ x y			bitwise xor x ^ y
| x y			bitwise or x
~ x			bitwise not, i.e.: inversion of x

Missing documentation

EX.LP.DOWNQ, EX.LP.REVQ, EX.M.CC.POUND, EX.M.N.POUND, EX.M.NO.POUND, EX.N.POUND, EX.NO.POUND, EX.POUND, G.XYP, G.XYP.X, G.XYP.Y, I2M.C.MINUS, I2M.C.PLUS, I2M.C.POUND, I2M.C.TTILDE, I2M.C.VTILDE, I2M.CC.OFF.POUND, I2M.CC.POUND, I2M.CC.SET.POUND, I2M.CC.SLEW.POUND, I2M.CCV.POUND, I2M.MAX.POUND, I2M.MIN.POUND, I2M.MUTE.POUND, I2M.N.POUND, I2M.NO.POUND, I2M.NRPN.OFF.POUND, I2M.NRPN.POUND, I2M.NRPN.SET.POUND, I2M.NRPN.SLEW.POUND, I2M.NT.POUND, I2M.POUND, I2M.Q.POUND, I2M.RAT.POUND, I2M.REP.POUND, I2M.S.POUND, I2M.SOLO.POUND, I2M.T.POUND, I2M.TEST, MI.SYM.DOLLAR, SYM.AMPERSAND.x3, SYM.AMPERSAND.x4, SYM.DOLLAR.F, SYM.DOLLAR.F1, SYM.DOLLAR.F2, SYM.DOLLAR.L, SYM.DOLLAR.L1, SYM.DOLLAR.L2, SYM.DOLLAR.S, SYM.DOLLAR.S1, SYM.DOLLAR.S2, SYM.LEFT.ANGLED.x3, SYM.PIPE.x3, SYM.PIPE.x4, SYM.RIGHT.ANGLED.x3, WS.S.POLY, WS.S.POLY.RESET, WS.SEL, WS1, WS2

Changelog

v5.0.0

- **FIX**: fix off-by-one error in `P.ROT` understanding of pattern length
- **FIX**: fix `CROW.Q3` calls `ii.self.query2` instead of `ii.self.query3`
- **FIX**: cache currently-running commands to avoid corruption during SCENE ops.
- **FIX**: delay when opening docs
- **FIX**: `PROB 100` would only execute 99.01% of the time.
- **FIX**: some `G.FDR` configurations caused incorrect rendering in grid visualizer
- **FIX**: fix `EX.LP` not returning correct values
- **FIX**: fix `QT.B` handling of negative voltage input
- **IMP**: scene load/save code refactor, add scene load/save tests
- **IMP**: fader ops now support up to four faderbanks
- **NEW**: new Disting EX ops: dual algorithms, `EX.M.N#`, `EX.M.NO#`, `EX.M.CC#`
- **FIX**: reset M timer when changing metro rate
- **NEW**: new Drum Ops: `DR.T`, `DR.V`, `DR.P`
- **NEW**: I2C2MIDI[41] ops
- **FIX**: fixes a transcription error in the SD drum helper patterns
- **NEW**: Ten new patterns for `DR.V` and optimised old patterns
- **FIX**: fix BPM rounding error
- **FIX**: support all line ending types for USB load
- **FIX**: fix `STATE` not accounting for `DEVICE.FLIP`
- **FIX**: fix MIDI IN ops channel number being off by 1
- **FIX**: improve `TR.P` accuracy
- **FIX**: fix `KILL` not stopping TR pulses in progress
- **NEW**: new op: `SCALE0` / `SCL0`
- **NEW**: new ops: `$F`, `$F1`, `$F2`, `$L`, `$L1`, `$L2`, `$S`, `$S1`, `$S2`, `I1`, `I2`, `FR`
- **NEW**: new op: `CV.GET`
- **NEW**: basic menu for reading/writing scenes when a USB stick is inserted
- **NEW**: new ops: `CV.CAL` and `CV.CAL.RESET` to calibrate CV outputs
- **FIX**: N.CS scales 7 & 8 were incorrectly swapped; make them consistent with N.S and docs
- **FIX**: libavr32 update: support CDC grid size detection (e.g. zero), increase HID message buffer
- **NEW**: new Disting EX ops: `EX.CH`, `EX.#`, `EX.N#`, `EX.NO#`
- **NEW**: apply VCV Rack compatibility patches, so branches off main can be used in both hardware and software
- **FIX**: update Disting EX looper ops to work with Disting EX firmware 1.23+
- **NEW**: new dual W/ ops: `W/.SEL`, `W/S.POLY`, `W/S.POLY.RESET`, `W/1`, `W/2`
- **NEW**: split cheatsheets into separate PDFs for core ops and i2c

[41] https://github.com/attowatt/i2c2midi

v4.0.0

- **FIX**: `LAST SCRIPT` in live mode gives time since init script was run
- **FIX**: negative pattern values are properly read from USB
- **FIX**: delay when navigating to sections in docs
- **NEW**: generic i2c ops: `IIA`, `IIS..`, `IIQ..`, `IIB..`
- **NEW**: exponential delay operator `DEL.G`
- **NEW**: binary and hex format for numbers: `B...`, `X...`
- **NEW**: Disting EX ops
- **FIX**: `LAST n` is broken for script 1
- **NEW**: bitmasked delay and quantize: `DEL.B..`, `QT.B..`, `QT.BX..`
- **NEW**: scale and chord quantize: `QT.S..`, `QT.CS..`
- **NEW**: bit toggle OP: `BTOG..`
- **NEW**: volts to semitones helper OP: `VN..`
- **IMP**: DELAY_SIZE increased to 64 from 16
- **FIX**: scale degree arguments 1-indexed: `N.S`, `N.CS`
- **NEW**: Just Friends 4.0 OPs and dual JF OPs
- **NEW**: binary scale ops `N.B` and `N.BX`
- **NEW**: reverse binary for numbers: `R...`
- **NEW**: reverse binary OP: `BREV`
- **NEW**: `ES.CV` read earthsea CV values
- **NEW**: added setter for R, sets R.MIN and R.MAX to same value, allowing R to be used as variable
- **NEW**: v/oct to hz/v conversion op: `HZ`
- **FIX**: W/2.0 ops added
- **NEW**: W/2.0 ops documentation
- **NEW**: `><`, `<>`, `>=<` and `<=>` OPs, checks if value is within or outside of range
- **IMP**: new powerful Q OPs
- **IMP**: Improved line editing movement (forward/backward by word skips intervening space).
- **NEW**: Delete to end of word command `alt-d` added.
- **NEW**: new op: `SCENE.P`
- **NEW**: new multi-logic OPs `AND3`, `AND4`, `OR3` and `OR4` with aliases `&&&`, `&&&&`, `|||` and `||||`
- **NEW**: alias: `EV` for `EVERY`
- **NEW**: live mode dashboard
- **NEW**: ops to control live mode: `LIVE.OFF`, `LIVE.VARS`, `LIVE.GRID`, `LIVE.DASH`, `PRINT`
- **FIX**: `PN.ROT` parameters are swapped
- **FIX**: better rendering for fine grid faders
- **FIX**: logical operators should treat all non zero values as `true`, not just positive values
- **NEW**: crow ops
- **NEW**: `TI.PRM.CALIB` alias added (was already in the docs)
- **FIX**: `SCENE` would crash if parameter was out of bounds

v3.2.0

- **FIX**: improve DAC latency when using CV ops
- **NEW**: call metro / init with `SCRIPT 9` / `SCRIPT 10`
- **NEW**: forward (C-f or C-s) and reverse (C-r) search in help mode
- **NEW**: new ops: `LROT` (alias <<<), `RROT` (alias >>>)
- **NEW**: `LSH` and `RSH` shift the opposite direction when passed a negative shift amount
- **NEW**: new op: `SGN` (sign of argument)
- **NEW**: new kria remote op: `KR.DUR`
- **NEW**: new op: `NR` (binary math pattern generator)
- **NEW**: new ops: `N.S`, `N.C`, `N.CS` (use western scales and chords to get values from `N` table)
- **NEW**: new ops: `FADER.SCALE`, `FADER.CAL.MIN`, `FADER.CAL.MAX`, `FADER.CAL.RESET` for scaling 16n Faderbank values (aliases `FB.S`, `FB.C.MIN`, `FB.C.MAX`, `FB.C.R`)
- **NEW**: new Tracker mode keybinding `alt-[ ]` semitone up, down
- **NEW**: new Tracker mode keybinding `ctrl-[ ]` fifth up, down
- **NEW**: new Tracker mode keybinding `shift-[ ]` octave up, down
- **NEW**: new Tracker mode keybinding `alt-<0-9> <0-9>` semitones up (0=10, 1=11)
- **NEW**: new Tracker mode keybinding `shift-alt-<0-9> <0-9>` semitones down (0=10, 1=11)
- **FIX**: dim M in edit mode when metro inactive
- **NEW**: new pattern ops: `P.SHUF`, `PN.SHUF`, `P.REV`, `PN.REV`, `P.ROT`, `PN.ROT`
- **NEW**: new pattern mods: `P.MAP:`, `PN.MAP x:`

v3.1.0

- **NEW**: new op: `DEVICE.FLIP`
- **FIX**: some keyboards losing keystrokes[42]
- **NEW**: new op: `DEL.X`
- **NEW**: new op: `DEL.R`
- **IMP**: DELAY_SIZE increased to 16 from 8
- **NEW**: new variables: J & K local script variables
- **FIX**: metro rate not updated after `INIT.SCENE`[43]
- **NEW**: new ops: `SEED`, `R.SEED`, `TOSS.SEED`, `DRUNK.SEED`, `P.SEED`, `PROB.SEED`
- **NEW**: new op: `SCENE.G`
- **NEW**: new op: `SCRIPT.POL`, alias `$.POL`
- **NEW**: new ansible remote ops: `ANS.G`, `ANS.G.P`, `ANS.G.LED`, `ANS.A`, `ANS.A.LED`
- **NEW**: new kria remote ops: `KR.CUE`, `KR.PG`

[42] https://github.com/monome/teletype/issues/156
[43] https://github.com/monome/teletype/issues/174

v3.0.0

- **NEW**: grid integration / grid visualizer / grid control mode
- **NEW**: multiline copy/paste and editing
- **NEW**: new keybindings to move by words
- **NEW**: undo in script editing
- **NEW**: i2c support for ER-301
- **NEW**: i2c support for 16n Faderbank
- **NEW**: i2c support for Matrixarchate
- **NEW**: i2c support for W/
- **NEW**: new op: ?
- **NEW**: new ops: `P.MIN`, `PN.MIN`, `P.MAX`, `PN.MAX`, `P.RND`, `PN.RND`, `P.+`, `PN.+`, `P.-`, `PN.-`. `P.+W`, `PN.+W`, `P.-W`, `PN.-W`
- **NEW**: new Telex ops: `TO.CV.CALIB`, `TO.ENV`
- **NEW**: new Kria ops: `KR.CV`, `KR.MUTE`, `KR.TMUTE`, `KR.CLK`, `ME.CV`
- **NEW**: new aliases: `$`, `RND`, `RRND`, `WRP`, `SCL`
- **NEW**: telex, ansible, just friends, w/ added to the help screen
- **FIX**: i2c initialization delayed to account for ER-301 bootup
- **FIX**: last screen saved to flash
- **FIX**: knob jitter when loading/saving scenes reduced
- **FIX**: duplicate commands not added to history[44]
- **FIX**: `SCALE` precision improved
- **FIX**: `PARAM` set properly when used in the init script
- **FIX**: `PARAM` and `IN` won't reset to 0 after `INIT.DATA`
- **FIX**: `PN.HERE`, `P.POP`, `PN.POP` will update the tracker screen[45]
- **FIX**: `P.RM` was 1-based, now 0-based[46]
- **FIX**: `P.RM` / `PN.RM` will not change pattern length if deleting outside of length range[47]
- **FIX**: `JI` op fixed[48]
- **FIX**: `TIME` and `LAST` are now 1ms accurate[49]
- **FIX**: `RAND` / `RRAND` will properly work with large range values[50]
- **FIX**: `L .. 32767` won't freeze[51]
- **FIX**: `I` now accessible to child SCRIPTS

v2.2

- **NEW**: added a cheat sheet PDF
- **NEW**: new bitwise ops: &, |, ^, ~, BSET, BCLR, BGET

[44] https://github.com/monome/teletype/issues/99
[45] https://github.com/monome/teletype/issues/151
[46] https://github.com/monome/teletype/issues/149
[47] https://github.com/monome/teletype/issues/150
[48] https://llllllll.co/t/teletype-the-ji-op/10553
[49] https://github.com/monome/teletype/issues/144
[50] https://github.com/monome/teletype/issues/143
[51] https://github.com/monome/teletype/issues/148

- **NEW**: new ops `PARAM.SCALE min max` and `IN.SCALE min max` to add static scaling to inputs
- **NEW**: blanking screensaver after 90 minutes of keyboard inactivity, any key to wake
- **NEW**: new op: `CHAOS` chaotic sequence generator. Control with `CHAOS.ALG` and `CHAOS.R`
- **NEW**: new op family: `INIT`, to clear device state
- **NEW**: new ops: `R`, `R.MIN`, `R.MAX` programmable RNG
- **IMP**: profiling code (optional, dev feature)
- **IMP**: screen now redraws only lines that have changed
- **FIX**: multiply now saturates at limits, previous behaviour returned 0 at overflow
- **FIX**: entered values now saturate at int16 limits
- **FIX**: reduced flash memory consumption by not storing TEMP script
- **FIX**: `I` now carries across `DEL` commands
- **FIX**: removed TEMP script allocation in flash
- **FIX**: corrected functionality of JI op for 1volt/octave tuning and removed octave-wrapping behaviour (now returns exactly the entered ratio)
- **FIX**: reduced latency of `IN` op

v2.1

- **BREAKING**: the `I` variable is now scoped to the `L` loop, and does not exist outside of an execution context. Scripts using `I` as a general-purpose variable will be broken.
- **FIX**: `SCENE` will not run from `INIT` script during scene load.
- **NEW**: Tracker data entry overhaul. Type numbers, press enter to commit.
- **NEW**: new op: `BPM` to get milliseconds per beat in given BPM
- **NEW**: script lines can be disabled / enabled with ctrl-/
- **NEW**: shift-enter in scene write mode now inserts a line
- **NEW**: new ops: `LAST x` for the last time script `x` was called
- **NEW**: `SCRIPT` with no arguments gets the current script number.
- **FIX**: `AVG` and `Q.AVG` now round up properly
- **NEW**: new op: `BREAK` to stop the remainder of the script
- **NEW**: new mod: `W [condition]: [statement]` will execute `statement` as long as `condition` is true (up to an iteration limit).
- **NEW**: new mods: `EVERY x:`, `SKIP x:`, `OTHER:` to alternately execute or not execute a command.
- **NEW**: new op: `SYNC x` will synchronize all `EVERY` and `SKIP` line to the same step.
- **NEW**: new feature: @ - the turtle. Walks around the pattern grid. Many ops, see documentation.
- **OLD**: ctrl-F1 to F8 mute/unmute scripts.
- **NEW**: ctrl-F9 enables/disables METRO.
- **FIX**: recursive delay fix. Now you can `1: DEL 500: SCRIPT 1` for temporal recursion.
- **FIX**: `KILL` now clears TR output as well as disabling the METRO script.
- **FIX**: if / else conditions no longer transcend their script
- **IMP**: functional exectuion stack for `SCRIPT` operations

v2.0.1

- **FIX**: update IRQ masking which prevents screen glitches and crashing under heavy load

v2.0

- **BREAKING**: remove `II` op. Ops that required it will now work with out it. (e.g. `II MP.PRESET 1` will become just `MP.PRESET 1`)
- **BREAKING**: merge the `MUTE` and `UNMUTE` ops. Now `MUTE x` will return the mute status for trigger x (`0` is unmuted, 1 is muted), and `MUTE x y` will set the mute for trigger x (`y = 0` to unmute, `y = 1` to mute)
- **BREAKING**: remove unused Meadowphysics ops: `MP.SYNC`, `MP.MUTE`, `MP.UNMUTE`, `MP.FREEZE`, `MP.UNFREEZE`
- **BREAKING**: rename Ansible Meadowphysics ops to start with `ME`
- **NEW**: sub commands, use a `;` separator to run multiple commands on a single line, e.g. `X 1; Y 2`
- **NEW**: key bindings rewritten
- **NEW**: aliases: `+` for `ADD`, `-` for `SUB`, `*` for `MUL`, `/` for `DIV`, `%` for `MOD`, `<<` for `LSH`, `>>` for `RSH`, `==` for `EQ`, `!=` for `NE`, `<` for `LT`, `>` for `GT`, `<=` for `LTE`, `>=` for `GTE`, `!` for `EZ`, `&&` for `AND`, `||` for `OR`, `PRM` for `PARAM`, `TR.P` for `TR.PULSE`
- **NEW**: new ops: `LTE` (less than or equal), and `GTE` (greater than or equal)
- **NEW**: new pattern ops: `PN.L`, `PN.WRAP`, `PN.START`, `PN.END`, `PN.I`, `PN.HERE`, `PN.NEXT`, `PN.PREV`, `PN.INS`, `PN.RM`, `PN.PUSH` and `PN.POP`
- **NEW**: USB disk loading and saving works at any time
- **NEW**: `M` limited to setting the metronome speed to 25ms, added `M!` to allow setting the metronome at unsupported speeds as low as 2ms
- **NEW**: TELEX Aliases: `TO.TR.P` for `TO.TR.PULSE` (plus all sub-commands) and `TI.PRM` for `TI.PARAM` (plus all sub-commands)
- **NEW**: TELEX initialization commands: `TO.TR.INIT n`, `TO.CV.INIT n`, `TO.INIT x`, `TI.PARAM.INIT n`, `TI.IN.INIT n`, and `TI.INIT x`
- **IMP**: new Ragel parser backend
- **IMP**: script recursion enhanced, maximum recursion depth is 8, and self recursion is allowed
- **IMP**: removed the need to prefix `:` and `;` with a space, e.g. `IF X : TR.PULSE 1` becomes `IF X: TR.PULSE`
- **IMP**: `AND` and `OR` now work as boolean logic, rather than bitwise, `XOR` is an alias for `NE`
- **FIX**: divide by zero errors now explicitly return a 0 (e.g. `DIV 5 0` now returns 0 instead of -1), previously the behaviour was undefined and would crash the simulator
- **FIX**: numerous crashing bugs with text entry
- **FIX**: i2c bus crashes under high `M` times with external triggers
- **FIX**: `P.I` and `PN.I` no longer set values longer than allowed
- **FIX**: `VV` works correctly with negative values

v1.4.1

- **NEW**: added Ansible remote commands `LV.CV` and `CY.CV`
- **NEW**: Added TELEX Modules Support for the TXi and the TXo
- **NEW**: 75 New Operators Across the Two Modules
- **NEW**: Supports all basic Teletype functions (add `TI` and `TO` to the commands you already know)
- **NEW**: Extended functionality allows for additional capabilities for existing functions
- **NEW**: Experimental input operators add capabilities such as input range mapping and quantization
- **NEW**: Experimental output operators add oscillators, envelopes, independent metronomes, pulse dividing, etc.
- **NEW**: Full List of Methods Found and Maintained Here[52]

v1.2.1

- **NEW**: Just Friends ops: `JF.GOD`, `JF.MODE`, `JF.NOTE`, `JF.RMODE`, `JF.RUN`, `JF.SHIFT`, `JF.TICK`, `JF.TR`, `JF.TUNE`, `JF.VOX`, `JF.VTR`

v1.2

- **NEW**: Ansible support added to ops: `CV`, `CV.OFF`, `CV.SET`, `CV.SLEW`, `STATE`, `TR`, `TR.POL`, `TR.PULSE`, `TR.TIME`, `TR.TOG`
- **NEW**: `P.RM` will also return the value removed
- **NEW**: `ER` op
- **IMP**: a `TR.TIME` of 0 will disable the pulse
- **IMP**: `O.DIR` renamed to `O.INC`, it's the value by which `O` is *incremented* when it is accessed
- **IMP**: `IF`, `ELIF`, `ELSE` status is reset on each script run
- **IMP**: key repeat now works for all keypresses
- **FIX**: `FLIP` won't interfere with the value of `O`
- **FIX**: the `O` op now returns it's set value *before* updating itself
- **FIX**: the `DRUNK` op now returns it's set value *before* updating itself
- **FIX**: `P.START` and `P.END` were set to 1 when set with too large values, now are set to 63
- **FIX**: `CV.SLEW` is correctly initialised to 1 for all outputs
- **FIX**: several bugs where pattern length wasn't updated in track mode
- **FIX**: fixed `[` and `]` not updating values in track mode

v1.1

- **NEW**: USB flash drive read/write

[52] https://github.com/bpcmusic/telex/blob/master/commands.md

- **NEW**: SCRIPT op for scripted execution of other scripts!
- **NEW**: MUTE and UNMUTE ops for disabling trigger input
- **NEW**: hotkeys for MUTE toggle per input (meta-shift-number)
- **NEW**: screen indication in live mode for MUTE status
- **NEW**: SCALE op for scaling number from one range to another
- **NEW**: JI op just intonation helper
- **NEW**: STATE op to read current state of input triggers 1-8 (low/high = 0/1)
- **NEW**: keypad executes scripts (works for standalone USB keypads and full-sized keyboards)
- **NEW**: KILL op clears delays, stack, CV slews, pulses
- **NEW**: hotkey meta+ESC executes KILL
- **NEW**: ABS op absolute value, single argument
- **NEW**: FLIP op variable which changes state (0/1) on each read
- **NEW**: logic ops: AND, OR, XOR
- **NEW**: O ops: O.MIN, O.MAX, O.WRAP, O.DIR for counter range control
- **NEW**: DRUNK ops: DRUNK.MIN, DRUNK.MAX, DRUNK.WRAP for range control
- **NEW**: TR.POL specifies the polarity of TR.PULSE
- **NEW**: if powered down in tracker mode, will power up in tracker mode
- **IMP**: TR.PULSE retrigger behaviour now predictable
- **IMP**: mode switch keys more consistent (not constantly resetting to live mode)
- **FIX**: bug in command history in live mode
- **FIX**: EXP op now exists
- **FIX**: P and PN parse error
- **FIX**: possible crash on excess length line entry
- **FIX**: CV wrapping with negative CV.OFF values
- **FIX**: INIT script executed now on keyboardless scene recall
- **FIX**: Q.AVG overflow no more
- **FIX**: P.PUSH will fully fill a pattern
- **FIX**: CV.SET followed by slewed CV in one command works
- **FIX**: DEL 0 no longer voids command

v1.0

- Initial release