zpwr-midi-fx - Reference Manual

v0.3.0

MODULE REFERENCE

Modular MIDI effects — transform the note stream before it reaches an instrument.

2026-06-21

MenkeTechnologies

Contents

Overview

zpwr-midi-fx is a **modular MIDI-effects plugin** that transforms the note stream before it reaches an instrument. Where a normal MIDI effect gives you one fixed transform (an arpeggiator, say, with a few knobs), zpwr-midi-fx hands you the same free-routed patch graph as zpwr-fx — instantiated on a stream of note events instead of audio samples — so you wire note-transform modules into your own chains. Turn single keys into voiced chords, scale-lock for intelligent harmony, run notes through a polymetric step arpeggiator with Euclidean rhythm generation, split the keyboard, add probability and humanisation — in any order and combination you wire. It runs as VST3, AU, CLAP and Standalone on macOS, Linux and Windows, registering as a MIDI-effect where the host supports it.

The modular advantage for MIDI: order is yours (harmonise then arpeggiate, or arpeggiate then harmonise, are different and both available), transforms can be modulated (an LFO sweeping a transpose, velocity driving a chord type), and you can fan a note stream into parallel branches (bass one way, chords another) and merge them.

Core concepts

The note stream. Where zpwr-fx processes audio samples, zpwr-midi-fx processes a stream of note events — note-ons, note-offs, velocities and controllers. Each block reads notes coming in, transforms them (adds, removes, retimes, retunes, re-velocities), and passes notes out. Placed before an instrument, it rewrites what the instrument actually receives, so a single key can arrive as a strummed seventh chord locked to your scale.

Blocks. A dynamic set you add, remove and reorder. Each block has a module type, note-stream inputs and a scalar **Mod** input (any number of cables per input, summed), up to six parameters, and one output. The two outputs, **Out A** and **Out B**, merge to the plugin's MIDI out — so you can build two parallel chains and combine them.

Scalar sources. Alongside notes the graph carries scalar control signals — LFOs, envelopes, Random, the macro soft keys, and the performance/MPE controllers. Patch these into a block's **Mod** input or route them to a parameter to animate

a transform (an LFO slowly shifting a Transpose, velocity driving an Arp's gate, MPE slide steering harmony).

Topological evaluation, no hung notes. The graph is evaluated once per audio block in dependency order, so each module runs after the modules it reads. Note-offs are scheduled across audio-block boundaries, so even with delays, ratchets, probability and long echoes in the chain, every note that starts is guaranteed to stop — the engine tracks the pairing for you.

Getting started

1. Place the plugin before an instrument — in a MIDI-effect slot, or on a MIDI track routed to an instrument.
2. Press ⚡ **EZ WIRE** — it auto-wires MIDI In → your blocks → Out so a chain works immediately, before you touch a cable.
3. Press **+ ADD BLOCK** to add a Chord, Arp or Scale module, then drag output jacks onto input jacks to build your own note pipeline. Drag an output to a second input to fan the stream into two branches.
4. Every control has a hover tooltip; **double-click** a block for its parameters and modulation.

INIT unplugs every cable and modulation while keeping the blocks; 🗑 blanks the patch.

The patch model

The **Inputs** column exposes every patchable source as a jack — the note input, every scalar modulator (Random plus the performance / MPE controllers) and the soft keys — so anything usable in the mod matrix can be cabled directly into a block as well.

Cables are drawn between jacks and dragged to re-patch. Right-click a cable for its editor: a **Level** that scales note velocity on that connection (`0` mutes it, and the cable's brightness and width follow the level — a built-in way to A/B a branch without deleting it), a **Colour** swatch, and **Disconnect**. Level is a live tweak; colour is cosmetic and persists with the patch.

A simple chord-then-arp chain wires as `MIDI In → Chord → Arp → Out A → MIDI Out`. Because order is explicit, reversing it (`Arp → Chord`) is a different musical result — arpeggiate the single keys first, then voice each arpeggiated note into a chord — and you choose which you want by how you wire.

Module library

The library spans several families. A representative selection follows; the **Module Reference** lists every block with its inputs and parameters, generated from the live registry so it never drifts.

Harmony — turn one key into many notes, or bend pitch into key:

Module	What it does
Chord	one key → a voiced chord (165 types, inversion, spread, octave-double, transpose, strum)
Harmonize	stack up to three fixed intervals
Scale	snap every note onto the nearest in-key pitch (20 scales)
Transpose	shift notes by semitones (modulatable)
Octave	add octave-up / octave-down copies
Invert	melodic inversion — reflect notes around a pivot
Fold	octave-fold every note into a fixed [low, high] window

Sequencing & rhythm — generate and reshape timing:

Module	What it does
Arp	host-synced arpeggiator (9 play modes, divisions, gate, octaves, swing)
SeqEuclid	retrigger held notes on a Euclidean (Bjorklund) rhythm
SeqRatchet	split each note into N rapid retriggers
Echo	MIDI delay with feedback (decaying repeats)
Strum	spread simultaneous notes in time (up / down)
Quantize	snap note timing to a grid (rate + strength)
Random	clock-driven random-note generator over a range

Module	What it does
Ramp	velocity crescendo / decrescendo over N notes

Dynamics & feel — velocity and timing:

Module	What it does
Velocity	scale / offset note velocity (modulatable)
VelCurve	reshape velocity through a gamma curve
VelClip	clamp velocity into a [min, max] window
Accent	boost the velocity of every Nth note (downbeat)
Humanize	jitter timing and velocity for a played feel
NoteLength	force every note to a fixed gate length

Routing, probability & control — gate, split and steer:

Module	What it does
Chance	per-note probability gate
NoteFilter	pass only notes within a note + velocity range (key zone)
KeySwitch	keyboard split — one zone plays, the other holds back
Mono	collapse to monophonic with note priority (last / lowest / highest)
Latch	toggle-hold notes — sustain until re-pressed
FixedNote	force every note to one pitch (drum triggering)
Channel	remap the output MIDI channel
Unison	layer each note as N copies, optional MPE channel spread
Merge	combine two note streams
RandOctave	randomly shift notes by ± octaves (probability)
LFO / Env / SampleHold / Slew	scalar control sources for the mod matrix

A family of **cellular-automaton sequencers** (Game of Life, Brian's Brain, Langton's Ant) evolve patterns from simple rules for generative, ever-changing chains.

Harmony & rhythm primer

A little theory makes the harmony and rhythm modules far more powerful.

Chords and voicings. A chord is a set of notes stacked over a root — a major triad is the root, its major third and its fifth. **Inversions** rotate which note is on the bottom (smoother voice leading between chords); **spread** opens the voicing across octaves (lush, less muddy); **octave doubling** reinforces the root. The Chord module does all of this from a single key, so you can play a progression with one finger and let the voicing do the musical work.

Scales and keys. A scale is the set of pitches that "belong" in a key. The Scale module snaps every incoming note to the nearest scale tone, so a wrong note becomes a right one. Place it after your generators and harmonisers and nothing you produce can ever be out of key — which is why generative patches (Random → Scale) always sound musical.

Intervals and harmony. Harmonize stacks fixed intervals (a fifth above, an octave below) for parallel harmony; Invert reflects a melody around a pivot note for a mirror line; Transpose shifts everything by semitones. Combine them for counterpoint from a single played line.

Arpeggios. An arpeggio plays a chord's notes one at a time. The Arp module's **mode** sets the order (up, down, up-down, as-played, random, and more), the **division** sets the speed relative to the host tempo, the **gate** sets how long each note rings, and **octaves** extend the pattern across the keyboard. Hold a chord and the arp spells it out in time.

Euclidean rhythm. A Euclidean pattern distributes a number of hits as evenly as possible across a number of steps — 3 hits in 8 steps gives the familiar x..x..x. tresillo. SeqEuclid gates your notes on such a pattern, and because you set hits and steps independently of your loop length, two Euclidean parts of different lengths drift in and out of phase for endlessly evolving polymetric rhythms.

Velocity and feel. Real performances breathe. Humanize adds small random timing and velocity variations; Accent pushes the downbeat; VelCurve reshapes how hard you have to play for a given loudness; Strum spreads a chord's notes slightly in time like a guitar. A pinch of each turns a robotic sequence into something that feels played.

Example chains

- **Instant harmony** — Chord → Scale: play single keys; Chord voices them and Scale locks every resulting note to your key, so nothing is ever out, no matter what you press. Add **Strum** for a roll.
- **Generative pluck** — Random → Scale → Arp: Random makes notes over a range, Scale locks them to key, Arp orders them rhythmically — an endless in-key sequence. Modulate Random's range with an LFO for slowly shifting registers.
- **Polymetric arp** — Arp → SeqEuclid: set the Arp division and the Euclid step length differently so the two cycles drift against each other for evolving rhythms; add **Echo** for trailing repeats.
- **Humanised live keys** — Chord → Humanize → VelCurve: voiced chords with subtle timing/velocity jitter and a velocity curve matched to your controller, so a stiff performance feels played.
- **Bass + chords split** — KeySwitch into two branches: the low zone through Mono → Transpose (down an octave) for a bass line, the high zone through Chord for voiced chords — both hands from one keyboard, merged to one instrument.

Modulation

A dynamic list of routes, each mapping a **scalar source** → **any block parameter** with a depth. Sources:

- **Soft keys** — an expandable pool of host-automatable macros (16 active by default, **+** / **-** to add or remove, count saved with the patch).
- **LFO** and **Env** block outputs, and **Random**.
- **Performance controllers** — mod wheel, pitch bend, aftertouch, velocity, expression (CC11), sustain (CC64).
- **MPE** — per-note bend, pressure and slide (CC74), read from the most recently active note's channel, so an MPE controller animates the transforms expressively (slide → chord type, pressure → velocity).

The same sources feed each block's **Mod** input. Routing a source/destination rebuilds the graph; depth is a live, lock-free tweak, so you can ride a modulation amount in real time without glitching.

Stereo

⊞ **Stereo** mirrors every block, cable and mod into an independent right-channel chain, kept in sync as you edit while the two sides' knobs stay independent so you can dial width into the patch. 🔒 **Lock** keeps the mirrored knobs tracking the left channel — bidirectional and offset-preserving, so a width you dialled in isn't reset to identical sides. Locked clone blocks are dimmed.

Perform & macros

The **Perform** tab is a macros-and-pads view with no patching, for live play:

- **Preset Morph** — a 4-corner XY pad bilinearly interpolating four captured presets; 🎲 fills all four at random.
- **Orb** — the puck's angle picks one of eight random scenes, its distance scales intensity; 🎲 rolls new scenes, ⏺ records the gesture and ▶ loops it.
- **XY macro pads** — each drives a pair of soft keys, with per-pad **HOLD** / **SPRING**.
- **Macro knobs**, eight **Snapshots** of the macro surface, and a 🎲 **Randomize**.
- **On-screen keyboard** — global **Key** + **Scale** quantize and a **Chord** selector (Off / Oct / 5th / Maj / Min / Maj7 / Min7 / Sus4 / Power) stacking intervals on each played key, plus a global arp (mode / rate / latch). The global arp is a quick performance layer; the **Arp** module is a patchable block you can place anywhere and modulate, with its own latch — use the global one to jam, the module when the arp is part of the design.
- **MIDI In** toggles — **Program** and **Bank**, both on by default: an incoming Program Change switches presets, with Bank Select (CC0 MSB / CC32 LSB) captured for the next Program Change.

Presets

A factory bank ships with the plugin — Chord Arp, Strummed Chords, Euclidean Pulse, Fifth Harmonizer, Arp Echo, Random Walk, Step Melody, Tone Cluster, Jazz Voicing, MPE Spread and more — spanning arps, harmony, rhythm, generative and FX chains, so the factory set doubles as worked examples of the module families. Your own patches save and load from the **Presets** tab, and the whole patch round-trips with your host project.

Generative & performance patches

Ten patches to build, from one-finger players to self-running machines. Wire them in the order written.

1. One-finger pianist. Chord → Strum → Scale. Press single keys; they voice into chords, strum like a guitar, and lock to your key. A whole accompaniment from one finger.

2. Endless arp. Random → Scale → Arp → Echo. Random notes locked to key, arpeggiated in time, with trailing MIDI echoes. Modulate Random's range with an LFO for drifting registers.

3. Polymetric pulse. Arp → SeqEuclid. Set the Arp division to 1/16 and the Euclid pattern to 5-in-8; the two cycles phase against each other into an evolving groove. Add a second SeqEuclid of a different length on a parallel branch and Merge.

4. Living chords. Chord → Humanize → RandOctave (low probability). Voiced chords that breathe, with the occasional note jumping an octave for organic variation.

5. Bass & keys split. KeySwitch into two branches: low zone → Mono → Transpose (-12) for bass, high zone → Chord → Strum for keys. Merge to one instrument. Play a whole arrangement two-handed.

6. Probability sequencer. A held chord → Arp → Chance. The arp spells the chord; Chance drops notes at random for an ever-shifting pattern. Raise Chance for density, lower it for sparseness.

7. Ratchet builds. Arp → SeqRatchet with the ratchet count on a soft key. Automate the soft key up over a bar for accelerating drum-roll builds.

8. Cellular melody. A GameOfLife (or Brian's Brain) sequencer → Scale → Arp. The automaton evolves a pattern; Scale keeps it musical; Arp gives it rhythm. A melody that never repeats.

9. MPE expression rig. Chord with the mod matrix routing **MPE slide → chord type** and **MPE pressure → velocity**. Press harder for louder, slide for different voicings — expressive harmony from an MPE controller.

10. Drum pattern generator. Random (narrow range) → FixedNote → SeqEuclid → Accent. Random triggers forced to one drum pitch, gated on a Euclidean rhythm, with accents on the downbeat. Point it at a drum instrument for generative beats.

Ten more patches

11. Jazz comping. Chord (7ths/9ths, random inversions) → Strum → Humanize. Rich, voice-led chords with a played feel — one-finger jazz comping.

12. Octave unison lead. Octave (up + down) → Unison (2). Every note thickened across octaves and doubled — huge trance/EDM leads from a single line.

13. Trance gate chords. Chord → SeqEuclid (fast division). Hold a chord; the Euclidean gate chops it into a rhythmic, pulsing pad.

14. Call and response. Two branches off the input: one dry, one through Echo → Transpose (+12), Merged. Your line answers itself an octave up, a beat later.

15. Probability drums. Several parallel FixedNote → Chance branches at different pitches and probabilities, merged — a generative drum machine where each voice has its own hit chance.

16. Mode-locked solo. Scale (Dorian) after your input; every note you play is forced into the mode, so you can improvise freely and never hit a wrong note.

17. Strummed harp gliss. Chord (wide spread) → Strum (slow, up). A single key blooms into a rolled, harp-like gliss across octaves.

18. Random walk melody. Random (small range, slow clock) → Scale → Glide feel via a downstream instrument's portamento — a wandering, in-key melodic line.

19. Velocity swell. Ramp cycling velocity up over N notes → your instrument. Automatic crescendos on repeated notes; reverse for decrescendos.

20. MPE chord spread. Chord → Unison (channel spread). Each chord note lands on its own MPE channel, so per-note bends and pressure stay independent downstream — expressive, polyphonic MPE chords from one key.

Modes, chords & scales

The harmony modules are most powerful when you know what to feed them.

Scales and modes. A scale is a set of intervals from a root. The major scale and its **modes** — Ionian (major), Dorian, Phrygian, Lydian, Mixolydian, Aeolian (natural minor), Locrian — each have a distinct flavour (Dorian is minor-but-hopeful, Phrygian is dark and Spanish, Lydian is dreamy). The Scale module offers twenty scale/mode types; pick the one whose mood you want and every note routes into it.

Chord types. Triads (major, minor, diminished, augmented) are three notes; sevenths add a fourth for jazz colour (maj7 is lush, dominant 7 wants to resolve, min7 is smooth); extensions (9ths, 11ths, 13ths) pile on more. The Chord module's 165 types span all of these — choose by the colour you want under your single-key melody.

Voicings. The same chord sounds different depending on note order and spacing. **Inversions** change which note is lowest (smoother movement between chords); **spread** opens the notes across octaves (less muddy, more open); **drop voicings** lower one note an octave (a jazz-piano staple). The Chord module's inversion, spread and octave-double controls are your voicing palette.

Putting it together. A progression in a key: choose your Scale mode, set Chord to the colour (say min7), play single roots, and add Strum + Humanize for feel. The modules supply the theory; you supply the tune.

Building a generative system

A generative patch makes music on its own and never repeats. The recipe has four stages, and each maps to a family of modules:

1. **A source of notes.** Something has to produce events. Random makes notes over a range on a clock; the cellular-automaton sequencers (GameOfLife, Brian's Brain, Langton's Ant) evolve patterns from rules; or a held chord through Arp spells out notes in time. This stage decides what plays.
2. **A musical filter.** Raw randomness is rarely musical, so lock it to key: Scale snaps every note into your scale, Fold keeps it in a register, NoteFilter removes out-of-range notes. After this stage, nothing can sound wrong.
3. **A rhythmic shape.** Give it time and groove: Arp orders notes, SeqEuclid gates them on an even pulse, SeqRatchet adds rolls, Quantize tightens timing, Strum spreads chords. This stage decides when things play.
4. **Variation over time.** Keep it alive: Chance drops notes for ever-shifting density, RandOctave jumps registers occasionally, Humanize adds human feel, and the **mod matrix** lets an LFO slowly move a parameter (a range, a probability, a transpose) so the system drifts.

Chain one module from each stage — say GameOfLife → Scale → SeqEuclid → Chance — and you have a self-playing instrument. Modulate one parameter with a slow LFO and it evolves for hours without repeating. Two such chains, each on a different pitch range and merged, give an interplay between parts.

MPE in depth

MPE (MIDI Polyphonic Expression) gives every note its own continuous controllers — bend, pressure and slide — instead of one set shared across the keyboard. zpwr-midi-fx works with it both as an input and as a tool:

- **As modulation.** Per-note **bend**, **pressure** and **slide (CC74)** are sources in the mod matrix, read from the most recently active note. Route slide → Chord type for morphing harmony, pressure → Velocity for touch-louder, bend → Transpose for expressive pitch.
- **Creating MPE.** Unison can spread each note's copies across separate MPE channels, so a chord becomes per-note-addressable downstream — an instrument that supports MPE can then bend or press each chord note independently. This turns a normal keyboard performance into MPE-ready material.
- **Keeping it clean.** Because note-offs are scheduled safely across blocks, even dense MPE chords through arps and ratchets won't leave hanging notes.

If you have an MPE controller (a Roli, a Linnstrument, a touch surface), routing its slide and pressure into the harmony and velocity modules makes the transforms themselves expressive — you're not just playing notes, you're playing the chord voicings and dynamics with your fingers.

Working with your DAW

- **Placement.** The plugin must sit before the instrument — in a MIDI-effect slot on the instrument track, or on a MIDI track routed to the instrument. It rewrites the note stream the instrument receives.
- **Tempo sync.** Arp, SeqEuclid, Echo and the clocked sources follow the host tempo and transport, so patterns stay locked to your project.
- **Automation.** Soft keys are host-automatable — assign performance moves (arp rate, chord type, transpose, a probability) to them and draw automation, or play them live on the Perform tab where they record.

- **Presets via program change.** Step through your patches with MIDI Program Change for live set changes (toggle in the Perform controls).

Songwriting with MIDI effects

MIDI effects aren't just for live tricks — they're a songwriting tool that suggests ideas you wouldn't play by hand.

- **Find a progression.** Set a Scale to your key and play single notes through a Chord module, trying different chord types. The voicings the module produces will suggest progressions — let the tool play the harmony while you hunt for the root movement.
- **Generate a hook.** Run Random → Scale → Arp and listen; when a phrase catches your ear, freeze it by recording the MIDI output into your DAW and editing from there. The generator is a brainstorming partner, not the final word.
- **Build tension and release.** Automate an Arp division from 1/8 to 1/16 to 1/32 over a bar (via a soft key) for a build; drop a SeqRatchet count up for a fill; pull a Chance probability down to thin a busy section and back up for the drop.
- **Vary a repeated part.** A looped phrase gets boring; insert Humanize, a low-probability RandOctave, or a slow Transpose automation so the repeat is never identical.
- **Counterpoint from one line.** Split your melody to a second branch through Echo → Transpose (an octave or a fifth) and merge — instant two-part writing.
- **Capture and commit.** Once a generative or harmonised part is right, record the plugin's MIDI output to a clip and turn the effect off, so the part is fixed and editable like any other MIDI.

The workflow that works: jam with the generators and harmonisers to discover parts, then capture the good ones as plain MIDI and arrange them. The plugin is the most productive as an idea machine.

Tips & best practices

- Order matters. Put **Scale** after harmony modules so every generated note is locked to key, and put **Humanize** last so it isn't undone by a later quantiser.
- Use ⚡ **EZ WIRE** to chain quickly, then reorder by re-patching cables — reordering is musical, not just cosmetic.
- A cable **Level** of 0 mutes a connection without deleting it — the fastest way to A/B a module's contribution.
- Assign performance moves (arp rate, chord type, transpose) to **soft keys** so they automate and live on the Perform pads.
- If repeats pile up, lower **Echo** feedback — note-offs are always scheduled safely, but very long tails can overlap into dense clusters by design.

FAQ

No notes reach my instrument. Press ⚡ **EZ WIRE**, or check that a chain reaches **Out A** / **Out B** and that the plugin sits before the instrument in the signal path.

My chords are out of key. Add a **Scale** module at the end of the chain, set to your key and scale.

How is the global arp different from the Arp module? The Perform tab's global arp is a quick performance layer over everything; the **Arp** module is a patchable block you place in the graph and modulate, with its own independent latch.

Does it support MPE? Yes — per-note bend, pressure and slide are modulation sources, and **Unison** can spread copies across MPE channels for per-note expression downstream.

Which formats and OSes? VST3, AU, CLAP and Standalone on macOS, Linux and Windows (AU is macOS only; Windows ships VST3 + CLAP). It registers as a MIDI-effect category where the host supports it.

Glossary

- **Note stream** — the flow of note events the plugin transforms before the instrument.
- **Block / module** — one note-transform unit: note inputs, a scalar Mod input, up to six params, one output.
- **Out A / Out B** — the two outputs that merge to the MIDI out (build two branches and combine them).
- **Scalar source** — an LFO/Env/Random/controller/soft-key value driving the mod matrix.
- **MPE** — per-note expression (bend, pressure, slide) usable as a modulation source.
- **EZ Wire** — one-click auto-routing of MIDI In → blocks → Out.

Plugin Architecture

JUCE MIDI-effects plugin — the MIDI-domain companion to zpwr-fx, on the same shared patch-graph engine. Paid product.

Build & Position

Metric	Value	Notes
Language	C++	JUCE MIDI-effect plugin
Engine	LayeredEngine<PatchEngine<NoteStream>>	shared zpwr-patch-core graph instantiated on the note-event stream
Formats	VST3 · AU · CLAP · Standalone	CLAP via clap-juce-extensions; registers as a MIDI-effect category where the host supports it (Logic AU MIDI FX, CLAP note ports)
Targets	macOS arm64/x86_64 · Linux x86_64/aarch64 · Windows x64/arm64	cross-platform, cross-arch from day one (Windows builds VST3 + CLAP via scripts/build_win.ps1)
Build system	CMake	CMakeLists.txt at repo root
Modules	111 module types	registerMidiModules — arp/chord/scale/euclid/transform/humanize/…; +1240+ audio/synth in the stack = 1300+ blocks
Domain	MIDI note stream	transforms notes before they reach an instrument
Status	stable	patch graph + full 111-module note-stream library, UI + factory presets

Architecture

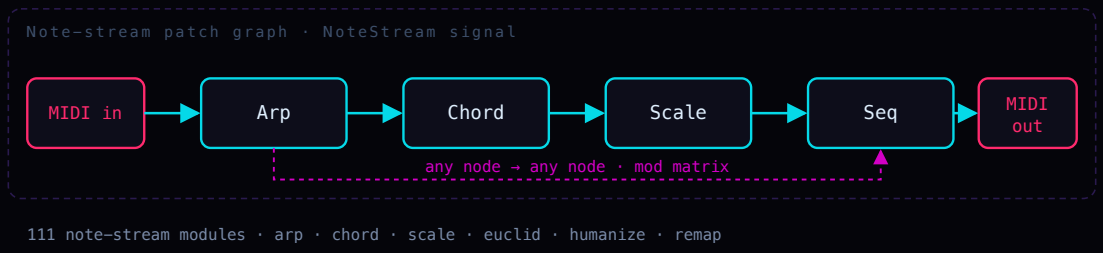


Figure 1: architecture diagram

The same zpwr-patch-core engine as zpwr-fx, instantiated on zmidi::NoteStream instead of float.

Not a fixed-slot rack: the same free-routed patch graph as zpwr-fx, instantiated on zmidi::NoteStream instead of float. Module types cover arpeggiation, chord generation, scale quantization, Euclidean and generative sequencing, velocity / timing humanize, and note remapping.

Catalog at a glance

111 blocks shipped by **zpwr-midi-fx** across **10 categories**, including 0 component-level analog-circuit models and 45 control / modulation (CV-badge) sources. This is the per-plugin subset of the shared zpwr-patch-core catalog.

Registry MIDI 111

Output jack Audio 0 MIDI 111 CV 0

By category (block count, largest first):

Reading this catalog (badge key)

An icon glyph precedes each block name; then its badges, then in: its input jacks and out: its output-jack type, then the description:

> [CAT] **category** - **AMP** Amp, **ANA** Analog, **CTL** Control, **DLY** Delay, **DST** Distortion, **DYN** Dynamics, **ENV** Envelope, **FLT** Filter, **FX** Effect, **HRM** Harmony, **MOD** Modulation, **OSC** Oscillator, **PIT** Pitch, **PRB** Probability, **REV** Reverb, **RTE** Routing, **SEQ** Sequencing, **SPC** Spectral, **STR** Stereo, **TUN** Tuning, **UTL** Utility, **VOI** Voicing.

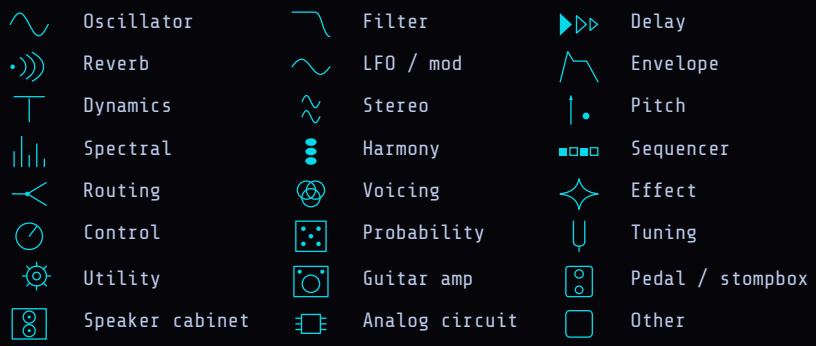
> [CV] a **control / modulation source** (outputs CV, not audio).

> [CIRCUIT] a **component-level analog-circuit model** (per-sample nodal / Newton solve).

> [FX] [SYNTH] [MIDI] which **plugin(s)** ship the block - zpwr-fx, zpwr-synth (gets the whole audio pack) and/or zpwr-midi-fx.

> **in:** the block's input jacks, and **out:** its output-jack signal type - out: Audio, out: MIDI (note stream) or out: CV (control / modulation).

> **icon:** the glyph before each block name shows its kind at a glance:



MIDI (zpwr-midi-fx) (111 blocks)

Control (5)

ATProc [CTL][CV][MIDI] in: Audio out: MIDI

Scales aftertouch, optionally converting it to a CC.

BendProc [CTL][CV][MIDI] in: Audio out: MIDI

Scales and/or inverts incoming pitch-bend.

CC2Note [CTL][CV][MIDI] in: Audio out: MIDI

A watched CC crossing a threshold triggers a note.

CCMap [CTL][CV][MIDI] in: Audio out: MIDI

Remaps a CC number and scales/offsets its value.

Note2CC [CTL][CV][MIDI] in: Audio out: MIDI

Emits a CC from each note (pitch or velocity).

Dynamics (10)

Accent [DYN][MIDI] in: Audio out: MIDI

Boosts velocity on every Nth note.

Humanize [DYN][MIDI] in: Audio out: MIDI

Adds random timing and velocity jitter for a human feel.

Ramp [DYN][MIDI] in: Audio out: MIDI

Velocity ramp over a run of notes, ascending or descending.

VelClip [DYN][MIDI] in: Audio out: MIDI

Clamps velocity between Min and Max.

VelCompress [DYN][MIDI] in: Audio out: MIDI

Velocity compressor: velocities above Threshold are pulled toward it by Ratio (1 = off, 8 = hard), narrowing the dynamic range smoothly instead of hard-clamping like VelClip.

VelCurve [DYN][MIDI] in: Audio out: MIDI

Reshapes velocity through a power curve.

VelInvert [DYN][MIDI] in: Audio out: MIDI

Flips the velocity scale around its midpoint (soft <-> loud) by Amount, so your accents become ghost notes and the ghosts become accents - a velocity mirror, blendable to taste.

VelKey [DYN][MIDI] in: Audio out: MIDI

Keyboard velocity tracking: scales each note's velocity by how far its pitch sits from a Center note, Tilt setting how strongly and which way - higher notes louder (or softer), the dynamic key-tracking of real instruments.

VelLength [DYN][MIDI] in: Audio out: MIDI

Note duration scales with velocity: hard notes ring for Max ms, soft notes for Min (set Max < Min to make accents staccato) - the played articulation where dynamics shape note length.

Velocity [DYN][MIDI] in: Audio out: MIDI

Scales and offsets note velocity, with a modulation amount.

Expression (3)

Fall [FX] [MIDI] in: Audio out: MIDI

Jazz fall-off: on release the pitch sweeps down by Depth over Time while the note rings, then the note releases and the bend snaps back to centre. The note-off is delayed to the end of the fall.

MidiVibrato [FX] [MIDI] in: Audio out: MIDI

A pitch-bend LFO that fades in after an Onset delay while any note is held - the delayed, played-in vibrato of a string or voice rather than an instant wobble. Rate in Hz, Depth as bend amount.

Scoop [FX] [MIDI] in: Audio out: MIDI

Bends up into each struck note from Depth below, resolving to pitch over Time - the vocal/brass scoop, a pitch-bend ramp that ends at centre so no bend is left hanging.

Harmony (21)

AutoInvert [HRM] [MIDI] in: Audio out: MIDI

Cycles the inversion of each struck chord: every trigger rotates which of the lowest notes jump an octave (by Rotate), so a repeated chord keeps climbing through its inversions - movement ChordVoicing's static shapes don't give.

Bassline [HRM] [MIDI] in: Audio out: MIDI

Follows the chord with a mono bass voice: the lowest held note dropped Octaves down, retriggered whenever the lowest note changes. Passes the original notes through and adds the bass beneath them.

Chord [HRM] [MIDI] in: Audio out: MIDI

Turns each incoming note into a chord (165 types) with inversion, spread, octave doubling and strum.

ChordLock [HRM] [MIDI] in: Audio out: MIDI

Quantizes every note to the nearest tone of a fixed chord (Root + Type), tighter than Scale's scale-quantize - locks an improvisation to a chord's notes.

ChordMem [HRM] [MIDI] in: Audio out: MIDI

Each key plays a stored chord shape (semitone intervals from node config).

ChordProg [HRM] [MIDI] in: Audio out: MIDI

Each note triggers the next chord in a stored root progression (node config).

ChordVoicing [HRM][MIDI] in: Audio out: MIDI

Re-voices each note into a jazz shape (open / shell / quartal / spread).

Counterpoint [HRM][MIDI] in: Audio out: MIDI

Adds a second voice in contrary motion - when the melody rises the counter voice falls and vice-versa - quantised to Root/Scale. Interval sets the starting separation.

Diatonic [HRM][MIDI] in: Audio out: MIDI

Transposes by scale DEGREES within Root/Scale (a third, a fifth…) instead of by raw semitones like Transpose - the shift always lands back in key. Mod Amt adds a modulatable degree offset.

DiatonicChord [HRM][MIDI] in: Audio out: MIDI

Auto-harmoniser: builds the in-key chord on every note by stacking diatonic thirds (triad, 7th…) along Root/Scale, so single-finger melodies become correct chords in the key.

Drone [HRM][MIDI] in: Audio out: MIDI

Doubles every note with a fixed pedal root.

Glissando [HRM][MIDI] in: Audio out: MIDI

Fills a stepwise run between consecutive notes.

Harmonize [HRM][MIDI] in: Audio out: MIDI

Adds up to three fixed-interval harmony voices to each note.

Invert [HRM][MIDI] in: Audio out: MIDI

Mirrors pitch around a pivot note.

MidiCluster [HRM][MIDI] in: Audio out: MIDI

Adds symmetric tone-cluster voices above and below each note.

MidiFold [HRM][MIDI] in: Audio out: MIDI

Folds out-of-range notes back inside a Low..High window.

NeoRiemann [HRM][MIDI] in: Audio out: MIDI

Neo-Riemannian triad transforms: builds a major/minor triad on each note and applies P (parallel), L (leading-tone) or R (relative) - the maximally-smooth chord moves of late-Romantic and film harmony, each sharing two common tones with the input.

Octave [HRM][MIDI] in: Audio out: MIDI

Doubles each note up and/or down by whole octaves.

Scale [HRM][MIDI] in: Audio out: MIDI

Quantizes incoming notes to the nearest degree of the chosen root and scale.

Transpose [HRM][MIDI] in: Audio out: MIDI

Shifts every note by a fixed semitone amount plus a modulatable offset.

VoiceLead [HRM][MIDI] in: Audio out: MIDI

Smooth voice leading: places each note's pitch class in the octave nearest the previous output, so a line or chord progression moves by the smallest interval. Range caps the leap; past it the original octave is kept.

Modulation (13)

CCGlide [MOD][MIDI] in: Audio out: MIDI

Portamento for a CC: smooths the watched CC toward each newly received value over Time, emitting interpolated CC each block - de-zippers a stepped controller, where MidiSlew only glides the internal scalar.

CCLFO [MOD][MIDI] in: Audio out: MIDI

Continuous CC LFO generator (sine/triangle/saw/square) emitted on a chosen CC at a free Rate - automates a synth parameter over MIDI, where MidiLFO drives only the internal mod-matrix scalar.

CCSeq [MOD][MIDI] in: Audio out: MIDI

Tempo-synced step CC sequencer: one CC value per synced step cycling an 8-step shape (overridable via the node's config list) - a modulation sequencer in the CC domain, where StepSeqMidi sequences notes.

Generative [MOD][MIDI] in: Audio out: MIDI

Probabilistic melodic random walk synced to a rate.

MidiEnv [MOD][MIDI] in: Audio out: MIDI

ADSR envelope triggered by the note gate, on the scalar control output.

MidiLFO [MOD][MIDI] in: Audio out: MIDI

Low-frequency modulation source (sine/tri/saw/square) on the scalar control output.

MidiRandom [MOD][MIDI] in: Audio out: MIDI

Generates random notes within a pitch range at a synced rate.

MidiSampleHold [MOD] [MIDI] in: Audio out: MIDI

Samples and holds the incoming control value on each note.

MidiSlew [MOD] [MIDI] in: Audio out: MIDI

Smooths (glides) the control value over a set time.

PitchDrift [MOD] [MIDI] in: Audio out: MIDI

Nudges each note's pitch by a small random detune up to +/-Cents (per-note, polyphonic) - the gentle oscillator instability of analog gear, applied to pitch where Humanize only scatters timing and velocity.

PressSwell [MOD] [MIDI] in: Audio out: MIDI

Generates a channel-pressure (aftertouch) swell that rises to Max over Rise while any note is held, snapping back to zero on release - an automatic crescendo for pads and strings, where ATProc only transforms incoming aftertouch.

RandOctave [MOD] [MIDI] in: Audio out: MIDI

Randomly shifts notes by up to +/- Range octaves with a chance.

VelToCC [MOD] [MIDI] in: Audio out: MIDI

Turns each note's velocity into a CC value emitted just before the note, so a synth's filter or level opens with how hard you played - dynamics routed to expression, where Note2CC sends pitch.

Probability (1)

Chance [PRB] [CV] [MIDI] in: Audio out: MIDI

Probabilistic gate: lets each note through with the set probability.

Routing (11)

Channel [RTE] [MIDI] in: Audio out: MIDI

Reassigns notes to a fixed MIDI channel.

FixedNote [RTE] [MIDI] in: Audio out: MIDI

Forces every note to a single fixed pitch (drum / trigger).

KeySwitch [RTE] [MIDI] in: Audio out: MIDI

Splits the keyboard at a note; one zone passes, the other is filtered.

Merge [RTE][MIDI] in: Audio out: MIDI

Passes both note-stream inputs through, merged into one stream.

MidiLatch [RTE][MIDI] in: Audio out: MIDI

Holds notes after release until the next note arrives.

NoteFilter [RTE][MIDI] in: Audio out: MIDI

Passes only notes inside a key and velocity window.

NoteLimit [RTE][MIDI] in: Audio out: MIDI

Polyphony cap: never lets more than Max notes sound at once, stealing the oldest (FIFO note-off) when a new note would exceed the limit - tames dense chords and runaway generators.

PolyChan [RTE][MIDI] in: Audio out: MIDI

Round-robins each new note to the next of Voices MIDI channels from Base, spreading a polyphonic part across a multitimbral rig - one note per channel, the classic voice-allocation split (distinct from MPEConvert's per-note MPE channels).

Split [RTE][MIDI] in: Audio out: MIDI

Keyboard split: notes below the split point transpose by Low, the rest by High.

Thin [RTE][MIDI] in: Audio out: MIDI

Note-density limiter: drops any note-on arriving within MinGap ms of the last one passed, decluttering a busy stream or fast repeats - distinct from NoteLimit, which caps simultaneous polyphony. A dropped note's off is dropped too.

Transformer [RTE][MIDI] in: Audio out: MIDI

Rule: notes within a range are transposed and velocity-scaled.

Sequencing (39)

■□■□ Appoggiatura [SEQ][CV][MIDI] in: Audio out: MIDI

Leaning grace note: each struck note first sounds a long auxiliary (Interval away) that takes Time from the principal, which then sustains - the expressive long grace, unlike the quick Flam or Mordent.

■□■□ Arp [SEQ][CV][MIDI] in: Audio out: MIDI

Arpeggiator: cycles held notes through a pattern at a synced rate with gate, octaves, swing and step count. Latch holds the chord so it keeps arpeggiating after the keys are released (the next key starts a new chord).

■□■□ **Cascade** [SEQ][CV][MIDI] in: Audio out: MIDI

Each struck note fans out into a timed run of Steps notes climbing (or falling) the scale from it - an arpeggiated flourish fired per key, unlike Arp which cycles the whole held chord.

■□■□ **ChordArp** [SEQ][CV][MIDI] in: Audio out: MIDI

Cthulhu-style chord arpeggiator: each key builds a full chord (Type) and the result is arpeggiated upward across Octaves at a synced Rate - one finger plays a moving arpeggio of the whole chord. Releasing the key stops its run.

■□■□ **Drunk** [SEQ][CV][MIDI] in: Audio out: MIDI

Random-walk melodic sequencer: each synced Rate step nudges the pitch by a random +/-Step semitones, bounded within +/-Range of the lowest held note - Brownian melody anchored to the chord, unlike MidiRandom's memoryless picks.

■□■□ **Echo** [SEQ][CV][MIDI] in: Audio out: MIDI

MIDI delay: repeats notes at a synced time with velocity feedback decay.

■□■□ **EuclidMel** [SEQ][CV][MIDI] in: Audio out: MIDI

Euclidean melody: spreads Pulses evenly across Steps (Bjorklund) and, on each hit, advances a cursor through the held chord - the rhythm is Euclidean and the pitch walks the chord, unlike SeqEuclid which stabs all held notes per pulse.

■□■□ **Flam** [SEQ][CV][MIDI] in: Audio out: MIDI

Precedes every note with a quiet grace note, delaying the main hit so the grace leads it - the classic drum flam.

■□■□ **GateArp** [SEQ][CV][MIDI] in: Audio out: MIDI

Cthulhu's arp gate lane: walks the held chord upward across Octaves while the note length cycles a per-step staccato/legato pattern, so the rhythm breathes - short stabs then sustained notes. Gate scales the pattern (>1 overlaps into the next step).

■□■□ **MidiBrianBrain** [SEQ][CV][MIDI] in: Audio out: MIDI

Brian's Brain MIDI sequencer: an 8x8 three-state grid (ready / firing / dying) evolves under Brian's Brain rules - a ready cell ignites only with exactly two firing neighbours, firing cells pass to dying, dying cells reset. Each clock step plays a note for every firing cell in the next column (row maps to a scale degree from Root/Scale); a full 8-step sweep advances one generation. The refractory state keeps it restless and sparkly where Game of Life settles. Density sets the random fill. The MIDI-note counterpart to the BrianBrain CV block.

■□■□ **MidiGameOfLife** [SEQ][CV][MIDI] in: Audio out: MIDI

Conway's Game of Life MIDI sequencer: an 8x8 cell grid evolves under the B3/S23 rule; each clock step reads the next column and plays a note for every live cell (row maps to a scale degree from Root/Scale), and a full 8-step sweep advances one generation. Gliders, blinkers and still-lifes become ever-shifting melodic and harmonic patterns. Density sets the random fill when the grid (re)seeds. The MIDI-note counterpart to the GameOfLife CV block.

▪▫▪▫ **MidiLangton** [SEQ][CV][MIDI] in: Audio out: MIDI

Langton's Ant MIDI melody: a single turmite walks an 8x8 grid - turning right on a white cell, left on a black one, flipping each cell as it leaves, then stepping forward. Each clock advances the ant Steps cells and plays one note for its current row (mapped to a scale degree from Root/Scale), so the melody scribbles chaotically then locks into the famous periodic 'highway'. Emergent order from one trivial rule - a single moving agent, not a population.

▪▫▪▫ **MidiMarkov** [SEQ][CV][MIDI] in: Audio out: MIDI

Learns the note-to-note transition statistics of what you play (a 12x12 pitch-class chain) and, on each synced step, walks the chain to improvise a new line in the same style. Memory fades old transitions.

▪▫▪▫ **MidiQuantize** [SEQ][CV][MIDI] in: Audio out: MIDI

Snaps note starts to a synced grid by a strength amount.

▪▫▪▫ **MidiTranceGate** [SEQ][CV][MIDI] in: Audio out: MIDI

Chops the held notes with a 16-step on/off Pattern at a synced Rate: every 'on' step retriggers all held notes, gated to a fraction of the step. The rhythmic gate behind the trance sound - pattern-driven, where Roll is a fixed pulse.

▪▫▪▫ **MidiTuring** [SEQ][CV][MIDI] in: Audio out: MIDI

Turing-machine sequencer (Music Thing style): a circular shift register clocks once per step and the held notes fire when the read bit is 1. Change is the probability the recycled bit flips - 0 locks an evolving loop, 1 fully randomises, between = slowly mutating melodies.

▪▫▪▫ **Mordent** [SEQ][CV][MIDI] in: Audio out: MIDI

One-shot ornament: each struck note plays principal -> auxiliary -> principal at the attack, then sustains, the single-flick counterpart to the continuous Trill.

▪▫▪▫ **NoteLength** [SEQ][CV][MIDI] in: Audio out: MIDI

Forces a fixed note duration (gate length).

▪▫▪▫ **NoteOffDelay** [SEQ][CV][MIDI] in: Audio out: MIDI

Extends each note's gate by delaying its note-off.

▪▫▪▫ **OctaveArp** [SEQ][CV][MIDI] in: Audio out: MIDI

Cthulhu's arp octave lane: arpeggiates the held chord while a per-step octave-offset Pattern leaps the line up and down by whole octaves (alternating, staircase, bounce…) instead of climbing monotonically like StepArp.

▪▫▪▫ **PatternGate** [SEQ][CV][MIDI] in: Audio out: MIDI

Gates note-ons through a 16-step on/off pattern (bitmask).

▪▫▪▫ **Polymeter** [SEQ][CV][MIDI] in: Audio out: MIDI

Two step lanes of different lengths (LenA, LenB) clock together over the held notes, phasing against each other so the combined pattern only repeats after lcm(LenA,LenB) steps - instant polymetric movement from one chord.

▪▫▪▫ **RatchetArp** [SEQ][CV][MIDI] in: Audio out: MIDI

Cthulhu's arp repeat lane: walks the held chord upward across Octaves while a per-step pattern retriggers selected steps multiple times (capped by Max), packing rolls into the line. Unlike SeqRatchet's fixed count, the repeat count varies per step.

▪▫▪▫ **Ricochet** [SEQ][CV][MIDI] in: Audio out: MIDI

Accelerating bouncing retriggers with decaying velocity.

▪▫▪▫ **Roll** [SEQ][CV][MIDI] in: Audio out: MIDI

Drum roll / buzz: retriggers every held note at Rate for as long as it is held, each hit gated to a fraction of the interval.

▪▫▪▫ **SeqEuclid** [SEQ][CV][MIDI] in: Audio out: MIDI

Euclidean rhythm gate: spreads Pulses evenly across Steps (rotatable) at a synced rate.

▪▫▪▫ **SeqRatchet** [SEQ][CV][MIDI] in: Audio out: MIDI

Retriggers each note Count times within a synced step.

▪▫▪▫ **Shift** [SEQ][CV][MIDI] in: Audio out: MIDI

Delays every note by Delay ms (groove nudge / lay-back), pushing note-ons and offs back by the same amount so durations are preserved - the per-track timing offset of a groove sequencer.

▪▫▪▫ **SkipArp** [SEQ][CV][MIDI] in: Audio out: MIDI

Cthulhu's arp skip lane: arpeggiates the held chord upward across Octaves, but each step has a Skip probability of being dropped to a rest, thinning the pattern differently every cycle for generative movement.

▪▫▪▫ **Spiral** [SEQ][CV][MIDI] in: Audio out: MIDI

Transposing MIDI delay: each repeat is delayed by Time and shifted by Interval semitones with velocity decaying by Feedback, so one note spins off a climbing or falling staircase of echoes - unlike Echo, which repeats the same pitch.

▪▫▪▫ **StepArp** [SEQ][CV][MIDI] in: Audio out: MIDI

Cthulhu-style step arpeggiator: walks the held chord strictly upward across Octaves octaves at a synced Rate, accenting the first step of each cycle - the rolling, octave-jumping arp of Xfer Cthulhu's sequencer. Distinct from Arp's mode-cycling: this climbs the chord in order.

▪▫▪▫ **StepSeqMidi** [SEQ][CV][MIDI] in: Audio out: MIDI

Melodic step sequencer - semitone offsets (node config) clock the held root.

▪▫▪▫ **Strum** [SEQ][CV][MIDI] in: Audio out: MIDI

Spreads a chord's notes over time, up or down, like a guitar strum.

▪▫▪▫ **Swing** [SEQ][CV][MIDI] in: Audio out: MIDI

Pushes off-beat eighth-notes later for a swing/shuffle feel.

▪▫▪▫ **TransArp** [SEQ][CV][MIDI] in: Audio out: MIDI

Cthulhu's arp transpose lane: arpeggiates the held chord upward across Octaves while a per-step semitone Pattern (fifth pedals, triad arps, scalar runs) is added on top, so the line outlines a melodic contour, not just the chord tones.

▪▫▪▫ **Trill** [SEQ][CV][MIDI] in: Audio out: MIDI

Keyboard trill: while a note is held, alternates rapidly between it and the note Interval semitones above, at Rate.

▪▫▪▫ **Tuplet** [SEQ][CV][MIDI] in: Audio out: MIDI

Snaps note starts to a Tuplets-per-beat grid (3 = triplets, 5 = quintuplets) by Strength, for a triplet or odd-tuplet feel where MidiQuantize only snaps to the straight grid.

▪▫▪▫ **Turn** [SEQ][CV][MIDI] in: Audio out: MIDI

Gruppetto turn: a four-step ornament at the attack - upper auxiliary, principal, lower auxiliary, then the principal held. Time sets each step, Interval the neighbour distance.

▪▫▪▫ **VelArp** [SEQ][CV][MIDI] in: Audio out: MIDI

Cthulhu's arp velocity lane as a ramp: arpeggiates the held chord while velocity sweeps from From to To across Steps then resets, so every pass swells or fades without a per-step editor.

Tuning (3)

JustTune [TUN][MIDI] in: Audio out: MIDI

Adaptive just intonation: retunes each note toward its 5-limit JI ratio relative to Key via per-note detune (polyphonic), Amount blending from equal temperament to pure - the beatless thirds and fifths of real harmony.

Microtuner [TUN][MIDI] in: Audio out: MIDI

Stretched-octave microtuning via per-note pitch detune.

TuneTable [TUN][MIDI] in: Audio out: MIDI

Custom microtuning: detunes each note by a per-pitch-class cents offset from the node's 12-value config list (relative to Key) - meantone, Pythagorean, maqam or gamelan, any temperament. Amount scales the table.

Voicing (5)

MidiUnison [VOI][MIDI] in: Audio out: MIDI

Doubles each note into multiple voices, optionally spread across channels.

Mono [VOI][MIDI] in: Audio out: MIDI

Monophonic note priority (last / lowest / highest).

MPEConvert [VOI][MIDI] in: Audio out: MIDI

Assigns each note its own rotating channel for per-note MPE expression.

Slide [VOI][MIDI] in: Audio out: MIDI

Cthulhu's slide: mono legato glue. Picks one held note by Priority (last/low/high) and emits the new note-on before releasing the old one, so a portamento synth slurs between notes instead of retriggering.

Sustain [VOI][MIDI] in: Audio out: MIDI

Sustain-pedal simulation: while Hold is engaged, note-offs are captured so the notes keep sounding; when Hold drops, every held note releases at once. Hold is a parameter, so a CC, an LFO or automation can play the pedal.

Index (111 blocks, alphabetical)