# Matter Stack — Choreography Engine

Software Specification: data formats, scene-file schema, and Arduino serial protocol

Companion to The Matter Stack — Living Instrument (Art & Expressive-Medium Addendum) · Engineering spec v0.1 — July 2026

## 1. Overview and Design Principles

The choreography engine is the real-time host program that turns a choreography (scripted keyframes plus live room-sound) into synchronized setpoints for the two microcontrollers driving the physical layers. It runs on a host computer — reference implementation in Python — and speaks to an **Arduino Mega** (the 8×8 terrain grid via four PCA9685 boards) and an **Arduino Nano** (the TinyLev levitator) over two USB-serial links.

Four principles govern the whole design:

- **Host computes, controllers actuate.** All choreography, audio analysis, mapping, and safety arithmetic happen on the host. Firmware is deliberately dumb: parse a frame, validate it, apply setpoints, run PWM, and fail safe. No creative logic lives on an Arduino.
- **One clock, one frame.** A single monotonic engine clock timestamps everything; the scene player, the modulation layer, and both serial writers all sample the same time `t`. A single normalized coordinate frame (Section 3) ties matter, sound, and light together.
- **Match signal speed to actuator speed.** The terrain is slow (0.2–1 s), the bead is fast (tens of ms). The engine drives each from the part of the signal it can actually follow — slow envelopes to terrain, discrete onsets to the bead.
- **Fail safe by default.** Every link carries a heartbeat; loss of it forces the controller to a safe state (coils off, levitator to a defined idle). Thermal and current limits are enforced on the host and re-checked in firmware as a last resort.

## 2. Architecture and Data Flow

The engine is a set of modules feeding a fixed-rate control loop. The loop samples inputs, evaluates the active scene blended with live modulation into a single `FrameState`, then hands that state to per-layer compute stages and the serial writers.

Pipeline: `microphone → sound analyzer → (scene player ⊕ modulation) → FrameState → {terrain engine → Mega} · {bead choreographer → Nano} · {photonic renderer → projector}`. An optional depth-camera registration service corrects the photonic stage in v2.

### Module rates and roles:

| Module | Rate | Role |
|---|---|---|
| Engine control loop | `60 Hz` | Fixed-timestep scheduler; produces one FrameState per tick. |
| Sound analyzer | `~86 Hz` | 1024-sample FFT, 512 hop @ 44.1 kHz; emits the feature vector (§4.2). |

| | | |
|---|---|---|
| Scene player | `60 Hz` | Evaluates keyframe tracks of the active scene at time t; handles cues/transitions. |
| Modulation layer | `60 Hz` | Adds smoothed audio-feature contributions onto scripted base values (§6). |
| Terrain engine | `30 Hz` | Feature list → heightfield → per-coil duty; thermal budget + slew limit (§7). |
| Bead choreographer | `50 Hz` | Bead target → levitator phase/drive; onset-driven darts (§8). |
| Photonic renderer | `30–60 Hz` | Projection-maps color/intensity to cell coords; uses commanded bead pos as fiducial. |
| Serial writers (×2) | `30 / 50 Hz` | Encode and transmit frames; read ACK/STATUS; run link watchdogs (§9). |
| Safety supervisor | `60 Hz` | Global thermal/current budget, heartbeats, ESTOP, clean shutdown (§11). |

## 3. Coordinate Frames and Units

All spatial authoring is in a normalized **cell frame**: `x, y ∈ [0,1]` across the fluid cell, origin at the front-left corner, x to the right, y away from the viewer. The engine maps normalized coordinates to physical coil positions using the geometry in the config file, so scenes are resolution-independent and survive a 4×4 → 8×8 change unmodified.

| Quantity | Symbol / field | Type & range | Notes |
|---|---|---|---|
| Cell position | `x, y` | `float [0,1]` | Normalized over the cell; origin front-left. |
| Terrain height | `h(x,y)` | `float [0,1]` | 0 = flat, 1 = max stable displacement. Not duty; mapped per-coil in §7. |
| Coil duty | `duty` | `uint8 [0,255]` | Wire value to Mega; firmware scales to 12-bit PWM. |
| Bead height | `bead.z` | `float [0,1]` | Position along the levitator's vertical node axis (TinyLev v1). |
| Bead x,y (v2+) | `bead.x, bead.y` | `float [0,1]` | Reserved for a phased MATD array; ignored by TinyLev firmware. |
| Color | `rgb` | `uint8[3] 0–255` | sRGB. Authoring may use HSV; engine converts. |
| Intensity | `level` | `float [0,1]` | Photonic brightness multiplier. |
| Time | `t` | `float seconds` | Monotonic from engine start; double precision. |

## 4. Runtime Data Formats

### 4.1 FrameState — the per-tick payload

Every control tick produces one immutable FrameState. It is the single source of truth handed to the compute stages. Terrain is authored as a compact *feature list* (a few parametric bumps) rather than a dense grid — this matches the physics (§3.3 of the addendum: only ~8–12 separable features) and keeps scenes small. The terrain engine

rasterizes it to the 8×8 grid at runtime.

```
FrameState = {
  "t": 12.480,                      # engine time, seconds
  "terrain": {
    "features": [                   # 0..~12 parametric bumps
      {"x":0.50,"y":0.50,"amp":0.82,"sigma":0.14},
      {"x":0.20,"y":0.75,"amp":0.35,"sigma":0.10}
    ],
    "floor": 0.00                   # baseline height added everywhere
  },
  "bead":  {"z":0.66,"drive":0.90,"slew":0.40},   # slew: 0 slow .. 1 instant
  "light": {"mode":"wash","rgb":[220,120,40],"level":0.7,
            "targets":"all"},       # or per-feature color list
  "flags": {"estop":false}
}
```

### 4.2 Audio feature vector

The analyzer emits one feature vector per audio hop, each field normalized to [0,1] by a running auto-gain (slow AGC) so a quiet gallery and a loud crowd both use the full range. Definitions:

| Field | Definition | Speed / use |
|---|---|---|
| `rms` | Windowed RMS loudness, AGC-normalized. | Slow → light intensity |
| `band_low` | Energy 20–250 Hz (bass). | Slow → central terrain feature |
| `band_mid` | Energy 250 Hz–4 kHz. | Medium → rim terrain features |
| `band_high` | Energy 4–16 kHz. | Medium → sparkle / light hue |
| `centroid` | Spectral centroid, mapped log to [0,1] (“brightness”). | Medium → light hue |
| `flux` | Spectral flux (positive change between frames). | Fast → texture / agitation |
| `onset` | Boolean + strength[0,1] from a peak-picked flux detector. | Fast → bead dart |

## 5. Scene-File Schema

A choreography is authored as a JSON **piece**: metadata, a settings block, an optional global `modulation` section (live-sound bindings), and an ordered list of **scenes**. Each scene has a duration, a transition, three keyframed tracks (`terrain`, `bead`, `light`), and a `cue` describing how it advances. Keyframes carry a time (seconds, relative to scene start), a value, and an easing.

### 5.1 Field reference

| Path | Type | Meaning |
|---|---|---|
| `piece.name / author / bpm` | `str / str / num` | Metadata. bpm optional, enables beat-based cue timing. |
| `settings.loop` | `bool` | Restart the piece when the last scene ends. |
| `settings.idle` | `obj` | Generative fallback used on silence or between pieces. |
| `modulation[]` | `array` | Global live-sound bindings (§6). Applied on top of every scene. |

| | | |
|---|---|---|
| scenes[].id | str | Unique scene name; used by cue targets. |
| scenes[].duration | num (s) | Nominal length; a cue may extend or cut it. |
| scenes[].transition | obj | {"type":"crossfade","dur":1.5} blend from previous scene. |
| ...terrain[] | keyframe[] | Each value is a terrain feature-list snapshot (see §4.1). |
| ...bead[] | keyframe[] | Each value = {z, drive, slew}. |
| ...light[] | keyframe[] | Each value = {rgb\|hsv, level, mode, targets}. |
| keyframe.t / .v / .ease | num / obj / str | Time in scene, value, easing: linear\|ease\|hold\|cubic. |
| scenes[].cue | obj | Advance rule: by timer, operator key, or audio trigger (§5.3). |

### 5.2 Worked example — the “Landfall” piece

A complete, valid three-scene piece. Terrain features are abbreviated for space; a real file lists each keyframe's full feature array. The listing continues across the two blocks below.

```
{
  "piece": {"name":"Landfall","author":"CB","bpm":72},
  "settings": {
    "loop": true,
    "idle": {"type":"perlin","amp":0.15,"speed":0.05}
  },
  "modulation": [
    {"src":"band_low","dst":"terrain.features[0].amp",
     "gain":0.6,"offset":0.0,"attack":0.25,"release":0.8,"clamp":[0,1]},
    {"src":"rms","dst":"light.level","gain":0.7,"offset":0.2,
     "attack":0.15,"release":0.6,"clamp":[0,1]},
    {"src":"onset","dst":"bead.impulse","gain":1.0,"mode":"event"}
  ],
```

```
  "scenes": [
    {
      "id":"calm_sea","duration":18.0,
      "transition":{"type":"crossfade","dur":2.0},
      "terrain":[
        {"t":0.0,"v":{"features":[{"x":0.5,"y":0.5,"amp":0.05,"sigma":0.30}],
                      "floor":0.0},"ease":"linear"}],
      "bead":[{"t":0.0,"v":{"z":0.25,"drive":0.8,"slew":0.2},"ease":"ease"}],
      "light":[{"t":0.0,"v":{"hsv":[210,0.5,0.6],"level":0.4,"mode":"wash",
                             "targets":"all"},"ease":"linear"}],
      "cue":{"type":"timer"}
    },
    {
      "id":"storm","duration":20.0,
      "transition":{"type":"crossfade","dur":1.0},
      "terrain":[
        {"t":0.0,"v":{"features":[{"x":0.5,"y":0.5,"amp":0.30,"sigma":0.16}],
                      "floor":0.05},"ease":"cubic"},
        {"t":8.0,"v":{"features":[{"x":0.5,"y":0.5,"amp":0.90,"sigma":0.12},
                                  {"x":0.25,"y":0.7,"amp":0.4,"sigma":0.10},
                                  {"x":0.78,"y":0.3,"amp":0.4,"sigma":0.10}],
                      "floor":0.1},"ease":"cubic"}],
      "bead":[{"t":0.0,"v":{"z":0.4,"drive":0.9,"slew":0.5},"ease":"ease"},
              {"t":8.0,"v":{"z":0.85,"drive":1.0,"slew":0.7},"ease":"ease"}],
      "light":[{"t":0.0,"v":{"hsv":[210,0.5,0.6],"level":0.5},"ease":"linear"},
               {"t":10.0,"v":{"hsv":[30,0.9,1.0],"level":0.9},"ease":"linear"}],
      "cue":{"type":"audio","when":"rms<0.15","hold":true,"timeout":30.0}
    },
```

```
    {
      "id":"stillness","duration":14.0,
      "transition":{"type":"crossfade","dur":3.0},
      "terrain":[
        {"t":0.0,"v":{"features":[{"x":0.5,"y":0.45,"amp":0.30,"sigma":0.22}],
                      "floor":0.05},"ease":"cubic"}],
      "bead":[{"t":0.0,"v":{"z":0.35,"drive":0.85,"slew":0.15},"ease":"ease"}],
      "light":[{"t":0.0,"v":{"hsv":[205,0.4,0.5],"level":0.3,"mode":"point"},
                "ease":"linear"}],
      "cue":{"type":"timer"}
    }
  ]
}
```

### 5.3 Cue types

| cue.type | Advances when | Fields |
|---|---|---|
| `timer` | Scene duration elapses. | – |
| `operator` | A key/OSC message is received (live performance). | `key` |
| `audio` | An expression over the feature vector becomes true. | `when, hold, timeout` |

## 6. Reactive Modulation Model

Live sound never replaces the scripted choreography — it *modulates* it. For each bound parameter the engine computes a final value as the scripted base (from the keyframe tracks) plus the sum of its bindings' contributions:

```
value(param, t) = base(param, t) + Σ  clamp( gain_i * curve_i(feat_i) + offset_i )
                                  i
smoothed feature (envelope follower), evaluated each tick dt:
    tau   = (x > y) ? attack : release          # asymmetric
    alpha = 1 - exp(-dt / tau)
    y    += (x - y) * alpha                      # y is the smoothed feature
```

Continuous bindings (e.g. `band_low → terrain.features[0].amp`) use the envelope follower so slow actuators track the *mood* of the sound, not its waveform; long release keeps the terrain from twitching. **Event** bindings (`mode:"event"`, e.g. `onset → bead.impulse`) fire on a discrete trigger and inject a fast, decaying impulse — this is what makes the bead feel rhythmic. Bindings are additive and order-independent; a per-parameter clamp keeps the total in range and protects the fluid's stable regime.

## 7. Terrain Engine — Features to Coil Duty

The terrain engine converts the feature list into 64 duty bytes in four stages, every 30 Hz frame:

**1. Rasterize.** Sample the height field on the 8×8 coil lattice as a sum of Gaussian bumps: $h(cx,cy) = \text{floor} + \Sigma\ amp_k \cdot \exp(-((cx-x_k)^2+(cy-y_k)^2) / (2 \cdot sigma_k^2))$, clamped to [0,1].

**2. Map to duty.** Apply each coil's calibrated height→duty curve (measured once in Build Part B and stored in config as a small lookup per coil). The relationship is non-linear; a per-coil curve corrects for coil-to-coil variation.

**3. Thermal budget.** If $\Sigma$ `duty` exceeds the global power cap, scale all duties by a single factor so the tallest features survive and the whole field dims together rather than clipping arbitrarily. A per-coil dwell integrator (an $I^2t$-style accumulator) forces any coil that has run hot to cool, overriding the setpoint if needed.

**4. Slew limit.** Clamp per-coil change per frame to a max Δduty so bumps travel smoothly and the fluid shears instead of splattering (the tear-speed measured in Build Part B sets the ceiling).

Output is a 64-byte array in row-major coil order, sent as a `TERRAIN_FRAME` (§9.2).

## 8. Bead Choreographer — Target to Levitator

For the TinyLev v1 the bead has one controllable axis: the standing-wave node it sits in, moved by the driver's phase-shift feature. The choreographer maps `bead.z ∈ [0,1]` to a phase setpoint `0..PHASE_MAX`, applies the requested slew (rate limit), and layers onset impulses on top: an `onset → bead.impulse` event adds a brief, fast offset that decays back to the scripted z, producing the darting “lead voice.” Drive level maps to the levitator amplitude (kept within the safe range from Build Part A). Bead x,y fields are carried through the pipeline but ignored by TinyLev firmware; they become active with the phased-array (MATD) upgrade, at which point only the choreographer's output stage and firmware change.

## 9. Serial Protocol

Two independent USB-serial links, each a framed binary protocol. Defaults: **Mega 250000 baud**, **Nano 115200 baud**, 8N1. Framing is **COBS** (Consistent Overhead Byte Stuffing) with `0x00` as the packet delimiter, so a payload can contain any byte value without collision.

### 9.1 Frame layout

Before COBS encoding, every packet is:

```
[ TYPE (1) | SEQ (1) | LEN (1) | PAYLOAD (LEN) | CRC16 (2, little-endian) ]
  CRC16 = CRC-16/CCITT-FALSE  (poly 0x1021, init 0xFFFF) over TYPE..PAYLOAD
  whole packet is then COBS-encoded and terminated with a 0x00 byte
  SEQ increments per packet; the receiver echoes it in ACK/STATUS
```

### 9.2 Host → Mega (terrain)

| Type | Name | Payload | Notes |
|---|---|---|---|
| 0x01 | TERRAIN_FRAME | 64 × uint8 | Row-major coil duty. Sent at 30 Hz. |
| 0x02 | SET_CONFIG | pwm_hz u16, duty_cap u8, max_hot u8 | PWM freq, global duty cap, max simultaneous hot coils. |
| 0x03 | ALL_OFF | — | Immediate: all coils to 0. ESTOP path. |
| 0x04 | PING | t_ms u32 | Heartbeat; expected every ≤100 ms. |
| 0x05 | HELLO | proto_ver u8 | Startup handshake; expects HELLO_ACK. |

### 9.3 Mega → Host

| Type | Name | Payload | Notes |
|---|---|---|---|
| 0x81 | STATUS | seq u8, flags u8, i_est u16(mA), t_c u8 | Echoes last SEQ; flags = ok/over-temp/over-current/underrun. |
| 0x82 | ERROR | code u8 | CRC fail, bad LEN, budget trip, etc. |
| 0x8F | HELLO_ACK | proto_ver u8, n_coils u8 | Capabilities reported at startup. |

### 9.4 Host → Nano (levitator)

| Type | Name | Payload | Notes |
|---|---|---|---|
| 0x10 | BEAD_TARGET | phase u16, drive u8, slew u8 | Node position, amplitude, rate limit. Sent at 50 Hz. |
| 0x11 | LEVITATOR_EN | on u8 | Enable/disable drive. Disable = safe idle. |
| 0x12 | SET_CONFIG | drive_max u8 | Cap amplitude to the safe range. |
| 0x04 | PING | t_ms u32 | Heartbeat; expected every ≤100 ms. |

### 9.5 Watchdog and example

Each firmware runs a receive watchdog: if no valid packet (any type) arrives within **150 ms**, it forces the safe state — Mega drops all coils to 0, Nano disables drive. The host therefore sends PING as a keepalive whenever it has no data frame to send. On host exit, the engine sends `ALL_OFF` and `LEVITATOR_EN 0` before closing the ports.

Example — a TERRAIN_FRAME with SEQ 42 and all-zero duty (pre-COBS):

```
TYPE=01 SEQ=2A LEN=40 PAYLOAD=00*64 CRC=<lo><hi>
 -> CRC-16/CCITT-FALSE over [01 2A 40 00...00]  (67 bytes)
 -> COBS-encode the whole packet, append 0x00 terminator, write to port
```

## 10. Timing, Threading, and Synchronization

The control loop uses a fixed-timestep accumulator (60 Hz) so behavior is deterministic regardless of host jitter. Audio capture runs in its own callback thread and publishes the latest feature vector through a lock-free single-slot buffer. Each serial link has its own writer thread consuming the newest Frame-derived packet (drop-oldest, never queue-and-lag) and its own reader thread parsing STATUS. Because both writers sample the same FrameState, the layers stay phase-locked to the shared clock.

| Stage | Typical latency | Consequence |
|---|---|---|
| Audio in → feature vector | ~12–15 ms | Onset detection is tight enough to feel rhythmic. |
| Engine tick | 16.7 ms | One-frame quantization of all outputs. |
| Serial + bead actuation | ~20–50 ms | Bead reads as responsive (the “lead voice”). |
| Serial + fluid response | ~0.2–1 s | Terrain reads the mood, by design (the “pad”). |

The bead doubles as the optical fiducial: because its commanded trajectory is known, the photonic renderer aligns to it with zero sensing latency. The terrain's true shape is emergent and, in v2, is measured by the depth camera to register light to real geometry.

## 11. Software Safety Supervisor

A dedicated supervisor runs at loop rate and owns the machine's safe state:

- **Global budget.** Maintains the running current/thermal estimate and enforces the duty cap and per-coil dwell integrator before frames leave the host (firmware re-checks as backstop).
- **Heartbeats.** Guarantees each link sees traffic within its watchdog window; escalates to ESTOP on repeated STATUS errors or link loss.
- **ESTOP path.** A single call drives `flags.estop`, which forces terrain to zero and the levitator to idle within one tick and latches until cleared.
- **Clean shutdown.** On exit or exception, sends ALL_OFF + LEVITATOR_EN 0, flushes, and closes ports — never leaves coils energized.

## 12. Config, Firmware Responsibilities, and Implementation Notes

### 12.1 Engine config (excerpt)

```
ports:   {mega: "/dev/ttyACM0", mega_baud: 250000,
          nano: "/dev/ttyUSB0", nano_baud: 115200}
grid:    {cols: 8, rows: 8, pitch_mm: 15, origin: "front_left"}
coil_cal:  "coil_curves.csv"        # per-coil height->duty lookup
thermal: {duty_cap: 190, max_hot: 20, dwell_ms: 4000}
rates:   {loop_hz: 60, terrain_hz: 30, bead_hz: 50}
audio:   {device: "default", sr: 44100, fft: 1024, hop: 512,
          bands: [250, 4000, 16000]}
projector: {homography: "cell_homography.json"}
```

### 12.2 Firmware does only this

Both Arduinos: read bytes, COBS-decode, verify CRC and LEN, apply setpoints (Mega → PCA9685 duty; Nano → phase/drive), service the receive watchdog, enforce a local last-resort duty/amplitude cap, and emit STATUS. No easing, no mapping, no scene logic — all of that is the host's job, which keeps firmware small, testable, and safe.

### 12.3 Reference stack and layout

Python reference: `pyserial` (links), `sounddevice + numpy` (capture/FFT), `cobs` (framing), `opencv` (projector homography, depth cam in v2). Suggested modules: `clock.py`, `scene.py`, `modulation.py`, `audio.py`, `terrain.py`, `bead.py`, `photonic.py`, `transport.py`, `safety.py`, `engine.py`, plus `pieces/*.json` and `config.yaml`.

### 12.4 v2 / V4 hooks

The schema and pipeline already carry what later stages need: `bead.x/y` for the MATD phased array; a registration service slot for the depth-camera closed loop; and, for the collaborative museum build (V4), the modulation layer generalizes from one microphone to multiple input sources by adding bindings whose `src` is a per-visitor control channel — one participant per terrain feature.