I can't download the ADR

You're right — canvas content isn't directly downloadable as a file. That's a tooling limitation, not a CLAP problem.

Here is the **complete ADR + contract document** in plain Markdown so you can paste it directly into:

Code/docs/ADR-000X-CLAP-Integration.md

or wherever you keep architecture decisions in joshband/LocusQ.

ADR-000X: CLAP Integration Contract for LocusQ

Date: 2026-02-21
Status: Proposed
Owners: LocusQ

Context

LocusQ is a quadraphonic spatial instrument/effect prioritizing:

- Deterministic DSP behavior
- No allocations in the audio thread
- Lock-free DSP→UI communication
- Explicit DSP↔️UI contract
- Regression-testable audio output

CLAP is an open plugin standard with:

- Stable ABI
- Extension-based capability model
- Optional per-voice modulation

- Optional note expression
- Sample-accurate event delivery

Hosts vary in which CLAP extensions they support. Therefore, LocusQ must treat CLAP as a **capability-negotiated runtime**, not a guaranteed feature surface.

Decision

LocusQ SHALL implement CLAP via a thin adapter layer that provides:

1. Explicit capability negotiation
2. Deterministic per-voice mapping
3. Sample-accurate modulation
4. Explicit expression routing

The spatial DSP engine SHALL remain format-agnostic.

Architectural Separation

Code[CLAP Host]
↓
[CLAP Adapter Layer]
↓
[LocusQ Spatial Engine]
↓
[Telemetry Ring → UI]

The spatial engine SHALL NOT depend on CLAP headers or host structures.

DSP_UI_CONTRACT_CLAP.md

1. Capability Negotiation

Requirements

- The adapter MUST query required CLAP extensions at initialization.

- The result MUST be stored in an immutable Capabilities struct.
- Runtime behavior MUST depend only on negotiated capabilities.
- Behavior MUST NOT depend on host name or version.
- Runtime mode MUST remain stable for the session.

Capabilities Structure

```
C++structCapabilities
{
bool hasParams = false;
bool hasNotePorts = false;
bool hasVoiceInfo = false;
bool hasNoteExpression = false;
bool hasPolyMod = false;
};
```

Runtime Modes

```
C++enumclassRuntimeMode : uint8_t
{
PolyVoice,
VoiceOnly,
GlobalOnly
};
```

Mode Selection Rule

- If hasVoiceInfo && (hasNoteExpression || hasPolyMod) → PolyVoice
- Else if hasVoiceInfo → VoiceOnly
- Else → GlobalOnly

Fallback Semantics

- VoiceOnly → per-voice state active, but no per-voice modulation.
- GlobalOnly → single logical voice; ignore voice IDs entirely.

---

2. Deterministic Per-Voice Mapping

Requirements

- Voice count MUST be bounded by kMaxVoices.
- No heap allocation in audio thread.
- Voice slot assignment MUST be deterministic.
- A slot MUST NOT be reused until release stage completes.

Voice Slot Model

```
C++constexprintkMaxVoices = 64;

enumclassVoiceStage : uint8_t
{
Off,
Active,
Release
};

structVoiceSlot
{
VoiceStage stage = VoiceStage::Off;
int32_t clapVoiceId = -1;
uint32_t age = 0;  // monotonic counter
float x = 0.f;
float y = 0.f;
float gain = 1.f;
};
```

Allocation Policy

1. First available Off slot (lowest index).
2. If none, steal oldest Release slot.
3. If none in Release, stealing Active is disallowed by default.

Tie-breaker MUST always be lowest index.

Lifetime Rules

- Note-on → stage = Active

- Note-off → stage = Release
- Slot returns to Off only when release envelope completes

---

3. Sample-Accurate Modulation

Requirements

- CLAP frame offsets MUST be honored.
- Modulation MUST be applied at exact sample positions.
- Base parameter storage MUST NOT be mutated by modulation.
- Effective value = base + modulation.

Modulation Event Structure

```
C++structModEvent
{
uint16_t frame;
uint8_t slot;
uint16_t paramId;
float value; // delta or absolute (must be consistent)
};
```

Processing Algorithm

For each block:

1. Parse CLAP events.
2. Translate into bounded ModEvent array.
3. Ensure deterministic ordering by frame.
4. Render audio in segments:

```
Codecursor = 0
for each event:
   render [cursor, event.frame)
   apply event
   cursor = event.frame
render [cursor, blockSize)
```

Overflow Policy

If modulation queue is full:

- MUST apply deterministic policy:
    - drop-oldest OR drop-newest
- Policy MUST be documented.
- Policy MUST be covered by QA tests.

---

4. Explicit Expression Routing

Requirements

- Expression routing MUST be explicit and serializable.
- No implicit mapping from expression to spatial parameters.
- Routing MUST occur at adapter boundary.
- Routing MUST be deterministic.

Spatial Targets

Minimum required:

```
C++enumclassSpatialTarget : uint16_t
{
X,
Y,
Gain
};
```

Optional (phase 2):

- Rotation
- Distance
- Diffusion

Expression Sources

Examples:

```
C++enumclassExprSource : uint16_t
{
PolyParam0,
NotePressure,
NoteTimbre,
NoteBrightness,
PitchBend
};
```

Routing Entry

```
C++structRoute
{
ExprSource source;
SpatialTarget target;
float amount;
float offset;
uint8_t curve; // linear, exp, etc.
};
```

Mapping Rule

```
Codemapped = curve(value) * amount + offset
```

Then emit `ModEvent` targeting resolved paramId.

Defaults

- No implicit expression routing.
- Routing MUST be opt-in.
- Factory presets MAY define curated routing profiles.

---

5. Telemetry & UI

Requirements

- DSP→UI telemetry MUST be lock-free.
- No allocation in audio thread.
- Overflow policy MUST be deterministic.
- UI MUST NOT be required for DSP correctness.

Telemetry SHOULD include:

- voice slot
- frame offset
- x
- y
- gain

---

6. QA Requirements

Mode Coverage

QA MUST include:

- PolyVoice mode
- VoiceOnly mode
- GlobalOnly mode

Determinism

QA MUST verify:

- Identical output for identical event streams
- Deterministic slot assignment under voice storms
- Deterministic modulation ordering

Golden Tests

Recommended cases:

- Static spatial positions
- Orbiting modulation
- Rapid per-voice LFO bursts
- Simultaneous note-on storms

- Adversarial modulation ordering

Open Decisions

- Delta vs absolute modulation semantics
- Active voice stealing policy
- Supported routing curves
- Constant-power normalization enforcement

If you'd like, I can now:

- Produce the matching `LocusQClapContract.h` header so implementation matches this spec exactly
- Add a CLAP-specific checklist for your CI pipeline
- Or generate a GitHub-ready version with issue links and TODO markers

Architecture only works if it's enforced.