violetaudio

dMix128 Integrator Manual

Version 1.2

---

1. Introduction & Design Philosophy

The **dMix128** is a network-centric digital mixing platform designed from the outset as a **control-first system**, rather than a console that exposes control as an afterthought. All signal processing, routing, and configuration parameters within the mixer are represented through a unified internal control model, enabling deterministic interaction from multiple external control systems simultaneously.

Unlike traditional mixers that prioritise a single control surface or UI metaphor, the dMix128 treats **all control interfaces as peers**. Web interfaces, automation systems, OSC controllers, Q-SYS control panels, and embedded monitoring tools all interact with the same underlying parameter space and observe the same system state.

This approach allows the dMix128 to be deployed in:

- Live sound environments with multiple operators
- Installed systems with building control integration
- Broadcast and automation workflows
- Hybrid live/install scenarios

without requiring different firmware, operating modes, or control hierarchies.

1.1 Core Design Principles

1.1.1 Deterministic State

Every controllable parameter in the mixer is represented by **exactly one internal key**, and that key has exactly one value at any given time. There is no per-client state, UI shadowing, or controller-specific interpretation once a command has entered the system.

This ensures:

- Predictable behaviour
- Elimination of state ambiguity
- Safe multi-client operation

1.1.2 Multi-Client Concurrency

The dMix128 is explicitly designed to support **multiple concurrent control clients**, potentially using different protocols, without conflict or priority inversion.

For example:

- A Q-SYS control panel may adjust input gains
- A TouchOSC tablet may control monitor sends
- An automation system may trigger snapshots
- A monitoring process may subscribe to system status

All of these actions coexist safely because they operate on the same internal state model.

1.1.3 Protocol Agnosticism

External protocols are treated purely as **translation layers**. Once an incoming command is translated into an internal key update, it is processed identically regardless of whether it originated from OSC, QRC, ECP, or VSKP.

This allows:

- Consistent behaviour across protocols
- Easier long-term maintenance
- Future protocol expansion without DSP changes

---

2. System Architecture Overview

The dMix128 control system is composed of four logical layers:

1. Mixer Engine
2. WebSocket Control Bridge
3. Central KeyStore
4. External Control Protocol Daemons

Each layer has a clearly defined role and communicates with adjacent layers using structured message flows.

2.1 Mixer Engine

The mixer engine performs all audio-related functions, including:

- Signal processing
- Routing
- DSP computation
- Snapshot execution

The mixer engine does **not** communicate directly with external control protocols. Instead, it exposes its parameters via a structured key/value interface to the control subsystem.

This separation allows the DSP engine and control system to evolve independently.

2.2 WebSocket Control Bridge

The WebSocket Control Bridge is the **sole communication path** between the mixer engine and the external control subsystem.

Key characteristics:

- Persistent TCP connection
- Bi-directional communication
- Event-driven updates
- Low-latency operation on local networks

2.3 Central KeyStore

The **KeyStore** is an in-memory mirror of all externally visible mixer parameters.

It stores:

- Numeric values (faders, gains, thresholds, frequencies)
- Boolean values (mute, solo, bypass)
- String values (names, labels)

The KeyStore serves as:

- The authoritative state reference for all protocols
- A change-detection mechanism

- A fan-out point for protocol-specific feedback

External protocols **never query the mixer engine directly**. All reads come from the KeyStore.

2.4 External Control Protocol Daemons

Each supported protocol is implemented as a daemon that:

1. Parses incoming protocol-specific messages
2. Translates them into internal keys
3. Formats feedback according to protocol expectations

Supported protocols:

- ECP (ASCII)
- QRC (JSON-RPC for Q-SYS)
- OSC (UDP and TCP)
- VSKP (Violet Simple Key Protocol)

2.5 End-to-End Message Flow

A typical control action follows this sequence:

1. External controller sends a command
2. Protocol daemon parses and translates it
3. Internal key is updated
4. KeyStore updates state
5. Mixer engine applies the change
6. Resulting updates are pushed back via WebSocket
7. KeyStore fans out feedback to other clients

This closed loop guarantees consistency across all connected systems.

---

3. Internal Key System (Authoritative Reference)

The internal key system is the foundation of all control and monitoring in the dMix128.

3.1 Key Format

Keys are dot-delimited strings forming a hierarchical namespace.

Examples:

- `i.0.mix`
- `i.0.mute`
- `i.0.eq.b2.gain`
- `i.0.aux.3.value`
- `a.2.mix`
- `m.mix`
- `v.1.mute`

3.2 Object Classes

- `i` — Input channels
- `a` — Aux buses
- `m` — Master bus
- `v` — VCAs
- `r` — FX returns
- `x` — Matrix outputs
- `s` — Subgroups

3.3 Indexing Rules

All internal indices are **zero-based**.

External protocols that use one-based numbering are translated automatically.

3.4 Data Types

- Numeric (double precision)
- Boolean (0.0 / 1.0)
- String

Boolean convention:

- `1.0` = active / engaged
- `0.0` = inactive

3.5 Action Keys

Some keys trigger actions rather than storing persistent state.

Examples:

- `load.snapshot.N`
- `save.snapshot.N`

3.6 Wildcards

- `*` matches one hierarchy level
- `**` matches recursively

Example:

```
i.*.mute
```

Recursive wildcards should be used cautiously in live systems.

---

4. Network, Transport & Ports

4.1 Port Table

Service	Protocol	Transport	Port
Mixer Core	WebSocket	TCP	80
ECP	ASCII	TCP	1702
VSKP	Text	TCP	1703
QRC	JSON-RPC	TCP	1710
OSC	UDP	UDP	10023

OSC	TCP	TCP	10024

4.2 TCP Behaviour

- Persistent connections
- Idle timeout (default 60s)
- Clients must reconnect on timeout

4.3 WebSocket Keepalive

Ping/pong keepalive ensures automatic reconnection and full resynchronisation via INIT.

4.4 OSC UDP vs TCP

UDP:

- Low latency
- No delivery guarantee

TCP:

- Reliable
- Framing auto-detection (SLIP / length-prefixed)

4.5 Security Model

The control system assumes a trusted network environment.

Recommended practices:

- Dedicated VLAN
- Firewall port restriction
- VPN for remote access
- No direct WAN exposure

---

5. State Synchronisation Model

State synchronisation ensures all connected clients observe a consistent mixer state.

5.1 INIT Phase

Upon connection, the control daemon issues an **INIT** request.

The mixer responds with:

- All exposed keys
- Current values
- Type information

No external feedback is sent until INIT completes successfully.

5.2 Real-Time Updates

After INIT:

- All mixer changes are pushed via WebSocket
- KeyStore updates internal state
- Protocol feedback is generated as required

5.3 Change Detection

The system tracks the last value sent to each client or Change Group to prevent redundant updates and feedback loops.

5.4 Protocol Feedback Models

- **VSKP** — push subscriptions
- **ECP / QRC** — change groups + polling
- **OSC** — query / reply

5.5 Normalised vs Native Values

Mapping functions convert between:

- Native units (dB, Hz)
- Normalised values (0.0–1.0)

Normalisation is explicit and protocol-controlled.

5.6 Multi-Client Behaviour

Any client may modify any key at any time. No ownership model exists.

---

6. Protocol Deep Dive – ECP

ECP is an ASCII-based control protocol.

Commands:

- `csv` — set native value
- `csp` — set normalised value
- `cg` — get value

Snapshots:

- `sl N` — load snapshot
- `ss N` — save snapshot

Change Groups provide efficient feedback.

---

7. Protocol Deep Dive – QRC (Q-SYS)

QRC uses JSON-RPC 2.0 framing.

Supports:

- Direct key access
- Mixer-oriented methods

- Range strings (`1-4,7`)
- Change Groups

Optimised for Q-SYS Designer workflows.

---

8. Protocol Deep Dive – OSC

Supports UDP and TCP.

Features:

- Standard mixer compatibility
- Index translation
- Mute inversion
- Normalised faders
- Query-reply semantics

Best suited for touch-based controllers.

---

9. Protocol Deep Dive – VSKP

VSKP is a lightweight, text-based protocol.

Operations:

- Query (`?`)
- Set (`=`)
- Increment (`+=`)
- Toggle (`!`)
- Subscribe (`+`)

Supports normalised keys (`%`) and wildcards.

Ideal for automation and monitoring.

---

10. Q-SYS Designer Practical Workflows

- Prefer QRC for new designs
- Use Change Groups
- Moderate polling rates (50–100 ms)
- Use snapshots for macro changes

---

11. OSC Controller Cookbook

- **StreamDeck** — toggles, snapshots
- **TouchOSC** — faders, monitor control
- **Mixing Station** — compatibility mode (note Mixing Station has a dMix128 version)

Avoid OSC for mission-critical monitoring.

---

12. Design Rules & Best Practices

- Prefer subscriptions over polling
- Avoid uncontrolled wildcard writes
- Normalise UI, not automation logic
- Assume multi-client operation

---

13. Diagnostics & Troubleshooting

Common issues:

- Missing INIT
- No subscriptions / change groups
- Normalisation mismatch
- Excessive polling

Appendix A – Quick Start for Programmers

VSKP:

```
+i.0.mix
i.0.mix=-12
```

OSC:

```
/ch/01/mix/fader 0.5
```

QRC:

```
{
  "jsonrpc": "2.0",
  "method": "Control.Set",
  "params": { "Name": "i.0.mix", "Value": -10 },
  "id": 1
}
```

END OF MANUAL TEXT