🔷 Q-SYS Control Protocol Overview

1. Native protocol

- Q-SYS communicates externally using **QRC (Q-SYS Remote Control protocol)** — this is a **WebSocket JSON API** on port **1710** by default.
- It also supports **third-party control** via:
  - **Lua scripting modules** (custom scripts in the Q-SYS Designer),
  - **TCP/UDP** network connections, and
  - **HTTP JSON API** (on Core 510i, Core Nano, Core 110f, etc., via `/jsonrpc`).

🔷 Integrating dMix128 with Q-SYS

Your **dMix128** already exposes an **OSC (UDP)** and/or **WebSocket JSON** control interface — both of which Q-SYS can access.

Option A — Simplest: OSC Integration (recommended)

Q-SYS supports **sending and receiving OSC** natively using the *"Control Script"* block or *"Lua script node"*.

Example Lua snippet inside a Q-SYS script block:

```
local udp = UdpSocket.New()
udp:Connect("192.168.1.120", 8000) -- dMix128 IP and OSC port

-- Set fader level to -10 dB (0.316 normalized)
udp:Send("/a.1.mix 0.316\n")

-- Mute channel 1
udp:Send("/a.1.mute 1\n")

-- Recall scene 3
udp:Send("/scene/load 3\n")
```

Your DMIX system would respond immediately because these messages match its OSC keymap.

💡 This allows direct fader, mute, and scene recall control from any Q-SYS control panel, button, or logic event.

Option B — WebSocket JSON Integration

If your dMix128 server exposes WebSocket (for browser GUI control), Q-SYS can also open a WebSocket connection and push JSON messages.

Example JSON (if using your Go backend API):

```
{
  "cmd": "set",
  "key": "a.1.mix",
  "value": 0.75
}
```

Q-SYS Lua WebSocket example:

```
local ws = WebSocketClient.New()
ws:Connect("ws://192.168.1.120:8080/ws")

ws:Send('{"cmd":"set","key":"a.1.mix","value":0.75}')
```

This is more flexible (allows scene names, stereo groups, metering feedback), but slightly more complex to configure.

Option C — Using QRC JSON (reverse direction)

If you ever want **DMIX128 to control Q-SYS**, you can send JSON-RPC calls to Q-SYS:

```
{
  "jsonrpc":"2.0",
  "method":"Component.Set",
  "params":{
    "Name":"GainControl1",
    "Controls":[{"Name":"Gain","Value":0.5}]
  },
  "id":"1"
}
```

Target URL:

```
ws://<qsys-ip>:1710
```

🔷 Recommended Integration Plan

Purpose	Connection	Port	Message Format	Example

Control dMix128 faders, mutes, scenes from Q-SYS	**OSC UDP**	8000	`/a.1.mix 0.75`	Straightforward
Two-way control / GUI sync	**WebSocket JSON**	8080	`{"cmd":"set","key":"a.1.mute","value":1}`	Bi-directional
Control Q-SYS from DMIX	**QRC JSON-RPC**	1710	Q-SYS JSON	`Component.Set`

If you tell me your **Q-SYS Core model** (e.g. Core 110f, Core Nano, or Core 510i) and how you want to connect (Ethernet vs local subnet),
I can write you a **ready-to-paste Lua script** for Q-SYS Designer that links directly to your dMix128's OSC or WebSocket layer.

Would you like me to generate that Lua script (with level, mute, and scene recall examples)?

We have made three ready-to-paste Q-SYS Lua scripts that talk to the dMix **WebSocket** layer (zero-based keys like `a.0.mix`, `a.0.mute`, plus scene recall). They're functionally identical (the Q-SYS Lua runtime is consistent across these cores), but are labeled per model for your project files.

- Download: dMix WebSocket for Core 110f
- Download: dMix WebSocket for Core Nano
- Download: dMix WebSocket for Core 510i

What's inside

- Connects to `ws://dmix.local:8080/ws` (change at the top of the file).
- Auto-reconnect with exponential backoff + keep-alive pings.
- Control map (you can extend it easily):
  - `Ch1 Level` ↔ `a.0.mix` (float 0..1)
  - `Ch1 Mute` ↔ `a.0.mute` (0/1)
  - `Ch2 Level` ↔ `a.1.mix`

- `Ch2 Mute` ↔ `a.1.mute`
- `Scene Number` (integer) + `Recall Scene` (button) → sends `{"cmd":"set","key":"scene/load","value":<n>}`

- Optional inbound feedback handler: if dMix emits updates like `{"evt":"update","key":"a.0.mix","value":0.51}`, the matching Q-SYS control updates.

Pins to add in Q-SYS Designer (exact names)

- `Ch1 Level` (Gain 0..1)
- `Ch1 Mute` (Toggle)
- `Ch2 Level` (Gain 0..1)
- `Ch2 Mute` (Toggle)
- `Scene Number` (Integer)
- `Recall Scene` (Button)
- *(Optional)* `Connect` (Button) to force a manual reconnect

Notes / tweaks

- dMix uses **zero-based** channel indices (`a.0.*` is Channel 1).
- To add more channels, duplicate the lines in the `MAP` table and increment the index (`a.2.mix`, `a.2.mute`, …).
- If your backend expects slightly different JSON, adjust `dmix_set()` and `recall_scene()` near the top.

Here is another example with 8 channel mixer with mute pan And the master level

Here are three Q-SYS Lua scripts mapped for **8 channels** with **Level, Mute, Pan**, plus a **Master Level**control, all talking to the dMix **WebSocket** layer.

- Core 110f — 8ch dMix WebSocket
- Core Nano — 8ch dMix WebSocket
- Core 510i — 8ch dMix WebSocket

Pin names to add in Q-SYS Designer (exact)

For each channel **1..8**:

- `ChN Level` (Gain 0..1)
- `ChN Mute` (Toggle)
- `ChN Pan` (Dial 0..1 — 0=L, 1=R; center at 0.5)

Global:

- `Master Level` (Gain 0..1)
- `Scene Number` (Integer)
- `Recall Scene` (Button)
- *(Optional)* `Connect` (Button)

Notes

- WebSocket URL defaults to `ws://dmix.local:8080/ws` at the top of each file — change if needed.
- Master fader key defaults to **`m.mix`**. If your backend uses something else (e.g., `lr.mix` or `main.mix`), change `MASTER_KEY` near the top.
- The scripts include auto-reconnect with exponential backoff, periodic pings, and optional feedback mapping if your dMix server emits updates like:

```
{"evt":"update","key":"a.0.mix","value":0.51}
```

Now let's add an emergency command that mutes channels 1 to 7 and make sure channel 8 is on and loud. Make this a different script.

Here is a separate "Emergency" script for each Core model. Pressing the **Emergency** button in Q-SYS will:

- **Mute channels 1–7** (`a.0..a.6.mute = 1`, levels set to `0.0`)
- **Force channel 8 ON and loud** (`a.7.mute = 0`, level set to `1.0` by default)

You also get an optional **Reset** button and a dial to tweak the **Emergency Loud Level** if you want.

Downloads:

- Core 110f — Emergency

- Core Nano — Emergency
- Core 510i — Emergency

Pins to add in Q-SYS Designer (exact names)

- `Emergency` (Button) — triggers the action
- `Reset` (Button) — optional; restores ch1–8 to 0.5 level and unmuted
- `Emergency Loud Level` (Dial 0..1) — optional; overrides the default 1.0
- `Connect` (Button) — optional; manual WebSocket connect

Notes

- The scripts connect to `ws://dmix.local:8080/ws` with keep-alive and auto-reconnect baked in.
- Keys assume your dMix mapping: `a.N.mute` (0/1) and `a.N.mix` (0..1), **zero-based** `N`.

We can also add a **Master Mute/duck** during emergency or a timed **latched release** (e.g., auto-reset after 30s)

To this download added **Master mute/duck** and a **timed latch (auto-reset after 30s)**, plus state snapshot/restore, and built separate scripts for each core.

Downloads

- Core 110f — Emergency (latched)
- Core Nano — Emergency (latched)
- Core 510i — Emergency (latched)

What this does

- On **Emergency** press:
  - **Ch1–7** → mute + set level to **0.0**
  - **Ch8** → unmute + set to **LOUD_LEVEL** (default **1.0**)
  - **Master** → either **mute** (`m.mute=1`) or **duck** (`m.mix=DUCK_LEVEL`, default **0.2**, and `m.mute=0`)
  - **Auto-reset** after **30 seconds** (configurable)
  - **Previous state** of Ch1–8 and Master (level/mute) is snapshotted and restored on reset.

Config you can tweak (at top of script)

- `DMIX_WS_URL` — default `ws://dmix.local:8080/ws`
- `LOUD_LEVEL` — how loud Ch8 should be during emergency (0..1)
- `DUCK_MODE` — `"duck"` or `"mute"`
- `DUCK_LEVEL` — master level when ducking (0..1)
- `MASTER_LEVEL_KEY` / `MASTER_MUTE_KEY` — change if your backend uses different keys
- `AUTO_RESET_SEC` — default **30**; set `0` to disable timed latch

Q-SYS control pins (names must match)

Required:

- `Emergency` (Button)

Optional/Recommended:

- `Reset` (Button) — manual release before the timer
- `Emergency Loud Level` (Dial 0..1) — overrides `LOUD_LEVEL` live
- `Master Duck Enable` (Toggle) — ON=duck, OFF=mute
- `Duck Level` (Dial 0..1) — sets duck level when ducking
- Mirrors for snapshot quality (optional but nice):
  - `Master Level` (Gain 0..1), `Master Mute` (Toggle)
  - `ChN Level` / `ChN Mute` for N=1..8
- `Connect` (Button) — manual WS connect

One bundle below with everything as a drop-in **Q-SYS Scriptable Controls** "**component kit"** with pin manifests and readme.

Download the bundle:
dmix_qsys_bundle_v1.zip

What's inside

- **8-ch Control scripts** (Core 110f, Core Nano, Core 510i): level/mute/pan for Ch1–8 + Master Level + Scene recall

- **Emergency (latched) scripts** (all three cores): mutes Ch1–7, forces Ch8 loud/on, Master **duck/mute**, **30s auto-reset**, snapshot/restore
- `manifest.json` — exact **pin lists** and defaults
- `README.md` — **step-by-step setup** in Q-SYS Designer

Quick install (summary)

1. Drop a **Scriptable Controls** block in your schematic
2. Set control count; **name pins exactly** per `manifest.json`
3. Paste the matching **Lua** into the block's script editor
4. Adjust `DMIX_WS_URL` if needed (`ws://dmix.local:8080/ws`)
5. Deploy to your Core

BSS / Harman HiQnet, We two practical paths depending on which devices you're talking to:

1) Soundweb London (BLU-80/BLU-160/BLU-100/BLU-3xx, etc.)

Use **London Direct Inject (DI)** over TCP. It's the intended third-party control path and is simpler than raw HiQnet.

- **Transport/ports**: TCP **1023** for DI control; Telnet on **23** (handy for testing). HiQnet discovery/control also uses **3804** TCP/UDP (but you don't need it for DI). help.harmanpro.com+1
- **Doc**: "Soundweb London Third-Party Control / London DI Kit" explains message framing and examples. help.harmanpro.com+1
- **Quick sanity tools**: Telnet login `bssaudio / monkey`, `help` shows commands. help.harmanpro.com

How to wire dMix → Soundweb (recommended)

Keep dMix on **OSC/WebSocket**, and add a small bridge service that:

1. Subscribes to your dMix events (e.g., `a.N.mix`, `a.N.mute`).
2. Opens a TCP socket to the Soundweb at **port 1023**.
3. Translates each event into the **DI message** for the specific parameter (gain/mute) using the object/parameter IDs you pull from your Audio Architect design. help.harmanpro.com+1

Tip: In **Audio Architect**, note the object instance and control IDs for the target blocks (gain, mute, router x-points). Those IDs map directly into the DI payloads you'll send. The Harman DI guide shows exact byte layouts and examples. BSS Networked Audio Systems

2) Generic HiQnet (beyond Soundweb: Crown amps, JBL processors, etc.)

You can speak **HiQnet** directly on **port 3804** (TCP/UDP). This is more involved but officially documented. help.harmanpro.com+2HiQnet Audio Architect+2

- **Doc**: "HiQnet Third-Party Programmer Documentation" (covers message formats, transports, command types, open/closed-loop control). HiQnet Audio Architect
- **Ports/firewall**: Ensure **3804 TCP/UDP** is open (plus FTP during updates on some devices). help.harmanpro.com+1
- **When to use**: If you need to control non-Soundweb HiQnet devices (e.g., Crown amps) from dMix without Audio Architect.

Which should you pick?

- **Controlling Soundweb London**: **Use London DI over TCP 1023**. It's straightforward and officially supported for third-party control. help.harmanpro.com+1
- **Controlling other HiQnet gear**: Implement **HiQnet on 3804** using the Programmer's Guide. Expect more framing/state management. HiQnet Audio Architect+1

Minimal bridge outline (concept)

- **Input**: dMix WebSocket JSON (e.g., `{"cmd":"set","key":"a.0.mix","value":0.73}`)
- **Output**: DI frame to Soundweb gain object for Ch1
- **Discovery**: None needed; you already know the BLU device IP.
- **Feedback**: Optionally poll or subscribe to meters/params via DI and push back to dMix as `{"evt":"update", ...}`.

(Exact DI bytes depend on your design's object/control IDs—grab them from **Audio Architect**; examples are in the DI Kit PDF.) BSS Networked Audio Systems

Network checklist

- Open **1023/TCP** to each Soundweb for DI; **23/TCP** optional for Telnet testing. help.harmanpro.com
- If you'll also use HiQnet discovery/control, open **3804 TCP/UDP** (and **21/49152-50152 TCP** if you do firmware/FTP). help.harmanpro.com+1

here's a **generic bridge** you can adapt for **BSS HiQnet / Soundweb London DI**, with the IP kept generic and the DI frames templated so you can drop in the exact bytes from your Audio Architect design.

Download: dmix_hiqnet_bridge_v1.zip
(Contains `main.go`, `config.example.json`, and a README.)

What it does

- Subscribes to your dMix **WebSocket** (`evt:"update"` messages like `{"key":"a.0.mix","value":0.73}`).
- For any mapped key, it builds a **DI/HiQnet frame** from a **hex template** and fires it over **TCP** to your BSS device.
- Auto-reconnects both the dMix WS and the BSS TCP session.

Configure it (keep IP generic or set later)

- In `config.example.json`, set:
  - `dmix_ws_url`: e.g. `ws://dmix.local:8080/ws`
  - `bss_host`: leave `"127.0.0.1"` for now; swap to your BLU device later
  - `bss_port`: `1023` (typical for London DI; change if you target raw HiQnet)
  - `mappings`: map dMix keys → **frame templates** (hex + tokens)

Template tokens supported:

- `{{VAL_F32_LE}}` → float32 little-endian
- `{{VAL_U8}}` → 0–255 byte

- `{{VAL_I16_BE}}`, `{{VAL_I16_LE}}` → 16-bit signed

Optional linear scaling per mapping:

```
"scale": {
  "mode": "linear",
  "in_min": 0.0, "in_max": 1.0,
  "out_min": -80.0, "out_max": 10.0
}
```

Example mapping (you'll replace headers/IDs per your design)

```
{
  "key": "a.0.mix",
  "frame_template": "AA 55 12 34 56 {{VAL_F32_LE}} CC",
  "scale":
{"mode":"linear","in_min":0.0,"in_max":1.0,"out_min":-
80.0,"out_max":10.0}
},
{
  "key": "a.0.mute",
  "frame_template": "AA 55 12 34 57 {{VAL_U8}} CC"
}
```

The `AA 55 ... CC` bytes are placeholders—fill them using the **London DI** or **HiQnet** docs for your object/parameter.

Build & run

```
cd dmix_hiqnet_bridge_v1
go build -o bridge main.go
./bridge -config ./config.example.json
```

If you want, I can also provide:

- A **DI checksum helper** token (if your frame needs it).
- A **BSS→dMix** poller path for **two-way sync** (read back params and push `evt:"update"` to dMix).
- A **pre-filled** config if you share your object/parameter IDs for the target gain/mute blocks.

Here's a breakdown of the **most popular installed-sound ecosystems** and **how dMix can talk to them**:

🎛️ 1. QSC Q-SYS

Market share: Extremely strong in corporate AV, stadiums, airports, and hospitality.
Protocol: QRC WebSocket JSON (port 1710) and **Lua scripting** inside Q-SYS Designer.
✅ **Status:** Already supported — your dMix WebSocket bridge + Lua scripts are ideal here.
➡️ *Next step:* Package an official Q-SYS "dMix Plugin" (.qplug) for Designer.

🧩 2. BSS Soundweb London / HiQnet

Market share: Strong in fixed installations, older but still widely deployed.
Protocol: London DI (TCP 1023) and **HiQnet (3804)**.
✅ **Status:** Bridge prototype done (our HiQnet bridge).
➡️ *Next step:* Add two-way feedback and publish parameter templates.

🔷 3. Biamp Tesira / Audia

Market share: Corporate boardrooms, government, education.
Protocol: Tesira Text Protocol (TTP) over TCP (Port 23).

- Simple human-readable ASCII commands like:

```
SET "Room1_Mic1_Gain" 0.55
```

- `GET "Room1_Mic1_Mute"`
-

✅ **Integration:** Trivial — dMix WebSocket ⇄ TCP bridge translating JSON↔TTP.
➡️ *High-value target*: Biamp's new AVB gear overlaps with your AES67 network.

⚙️ 4. Symetrix Radius / Prism / Edge

Market share: Mid-sized installations (houses of worship, education).
Protocol: SymNet Control Protocol (ASCII) over TCP (48631).

- Example: `SET 12 0.75\r\n` (where 12 = control ID).
  ✅ **Integration:** Very easy mapping to dMix keys (like we did for Q-SYS).
  ➡️ *Low-hanging fruit* for an early partnership.

🖥️ 5. Peavey MediaMatrix NION

Market share: Stadiums, large venues, cruise ships.
Protocol: NWare Text Control (Telnet-style ASCII), plus **SNMP**.
➡️ Bridge: Similar to Biamp — JSON⇄text translator.

🔊 6. Atlas IED BlueBridge / BluBlox (rebadged BSS)

Protocol: Same **London DI**, already covered.

🧠 7. Extron DSP & Control Processors

Market share: Education, corporate, government.
Protocol: SIS (Simple Instruction Set) ASCII via TCP/Serial.
Example:

```
1*Vol!   (set volume)
1V!      (query volume)
```

✅ Bridge: small dMix ⇄ SIS proxy for fader/mute levels.
➡️ Great for small-room control panels.

🌐 8. Crestron & AMX

Market share: Enterprise control systems.
Protocol: TCP/IP ASCII, **HTTP/JSON**, or **Serial** to third-party devices.
✅ Integration path: Crestron/AMX module sends HTTP or WebSocket to dMix directly (no bridge required).
➡️ *Big win* if you publish a Crestron SIMPL+ or Crestron HTML5 driver for dMix.

🧩 9. Dante-enabled DSPs (e.g. Powersoft, Lab Gruppen PLM+, Yamaha DME)

Market share: Large performance venues.
Protocols:

- **Dante Device Protocol** (for subscription control)
- **Yamaha DME**: proprietary TCP API
- **Powersoft Armonía+**: JSON API (port 5000)
  ✅ Integration: focus on **AES67/Dante interoperability** and gain/scene recall via JSON or OSC.

🧠 10. Allen & Heath dLive / Avantis (Install Mode)

Protocol: AHNet TCP/UDP JSON (undocumented but used by Director app).
➡️ *Experimental integration possible* for cross-scene sync in hybrid systems.

📊 11. Bose ControlSpace ESP / EX

Protocol: CSZ / ControlSpace Serial over Ethernet (CSP) — ASCII commands.
Example:

```
SET 1 Gain 0.25
```

✅ Straightforward for level/mute sync.

🧩 12. Harman BLU DA / Crown DCi / JBL Networked Amps

Protocol: HiQnet / Harman Audio Architect JSON & XML (port 3804).
✅ Already covered by your HiQnet bridge.

🔐 13. Rane HAL / RAD

Protocol: Rane HAL ASCII over TCP (4999).
Simple text commands, easy to map to JSON.

🧠 14. Shure Microflex / P300 / IntelliMix

Protocol: HTTP REST API + JSON, WebSocket updates for mutes, gains, meters.
✅ Super clean integration — dMix can mirror Shure mics directly over REST.

🔗 15. RTI / Control4 / Savant

Protocol: Typically **IP ASCII / HTTP JSON**; these act as the master.
✅ dMix can expose the same WebSocket JSON interface (you already have it).

🧱 Summary Table

Manufacturer / Platform	Protocol	Port	Integration Level
Q-SYS	QRC JSON (WebSocket)	1710	✅ Complete

BSS / HiQnet	DI / 3804	1023 / 3804	✅ Prototype
Biamp Tesira	TTP (ASCII)	23	🔶 Easy
Symetrix	SymNet (ASCII)	48631	🔶 Easy
Extron	SIS (ASCII)	varies	🔶 Easy
Crestron	HTTP / WebSocket	any	🔶 Easy
Bose	CSP ASCII	1024	🔶 Easy
Shure	REST / WebSocket	80 / 443	✅ Modern
Powersoft	JSON	5000	🔶 Modern
Rane	ASCII	4999	🔶 Easy

🔧 Recommendation: build these bridges next

1. **Biamp Tesira** (TCP 23) → easiest large-scale target.
2. **Symetrix SymNet** → quick win, same pattern as Biamp.
3. **Shure P300 / MXA** → modern JSON, great for corporate mic arrays.
4. **Extron SIS** → ties dMix into classrooms and meeting rooms.
5. **Crestron / AMX modules** → get integrators excited; publish on their driver stores.

-here's a **ready-to-use bundle** of minimal bridge templates for the key installed-sound ecosystems, all wired to your dMix WebSocket update stream and kept **generic** so you can drop in the exact command syntax for each platform.

Download: dmix_installs_bridge_samples_v1.zip

What's inside (folders)

1. **tesira_ttp/** – Biamp Tesira (TTP over TCP)

    - Sends ASCII like `SET "Mic1_Gain" {{VAL}}` (you fill in instance/control names).

2. **symetrix_symnet/** – Symetrix (TCP 48631)
    - ASCII like `SET <controlID> {{VAL}}` (use Composer IDs).
3. **extron_sis/** – Extron SIS (TCP/Serial)
    - Example volume/mute macros (device-specific).
4. **crestron_amx_gateway/** – HTTP helper gateway
    - Exposes `/set` endpoint so Crestron/AMX can call `GET /set?a.0.mute=1` → forwards to dMix WS.
5. **bose_csp/** – Bose ControlSpace (CSP ASCII)
    - ASCII like `SET 1 Gain {{VAL}}` (fill instance/param names).
6. **shure_rest/** – Shure (REST/JSON)
    - HTTP PUT/PATCH with a `json_template` you define (MXA, P300, IntelliMix).
7. **powersoft_json/** – Powersoft (generic JSON)
    - HTTP POSTs with scaling to dB; adapt to your Armonía+/API layout.
8. **rane_hal/** – Rane HAL (TCP 4999)
    - ASCII control strings (fill per HAL programmer's guide).
9. **common/** – Shared Go WebSocket helper.

Each sample includes:

- `main.go` – the tiny bridge
- `config.example.json` – set device IP/port + mappings
- `README.md` – per-target notes

Build & run (example)

```
cd tesira_ttp
```

```
cp config.example.json config.json  # edit for your IP,
mappings
go build -o bridge main.go
./bridge -config ./config.json
```

Mapping & scaling

- Map dMix keys (`a.N.mix`, `a.N.mute`, `a.N.pan`, `m.mix`, etc.) to each device's command via `command_template` or `json_template`.
- Optional **linear scaling** converts 0..1 → dB/percent/etc.

If you want, I can also:

- Pre-fill any folder with your exact **instance/control IDs** (once you share them),
- Add **two-way feedback** (poll device → push `evt:"update"` back to dMix),
- Combine multiple targets into a **single multiprotocol bridge** with per-device workers.

here's a **v2 bundle** that adds **two-way feedback** to each bridge so dMix reflects state changes made on the external DSP/amp systems.

Download: dmix_installs_bridge_samples_v2_twoway.zip

What's new (v2 highlights)

- Per-mapping **feedback pollers** (ASCII or REST) with configurable intervals.
- **Regex parsers** for ASCII replies (Biamp, Symetrix, Extron, Bose, Rane).
- **JSON-path extraction** for REST replies (Shure, Powersoft).
- Pushes **`{"evt":"update","key":"…","value":…}`** back to your dMix WS so the GUI/server stays in sync.
  (If your backend expects a different envelope, tweak `common/PushUpdate`.)

Included two-way templates

1. **Biamp Tesira (TTP)** — TCP (often 23).
   - Example feedback config:

```
"feedback": { "query": "GET \"Mic1_Gain\"",
"interval_ms": 1000, "regex": "([-0-9.]+)" }
```

-

2. **Symetrix SymNet** — TCP 48631, `SET/GET <controlID> ....`
3. **Extron SIS** — TCP/serial, device-specific macros; sample query+regex provided.
4. **Bose ControlSpace (CSP ASCII)** — TCP ASCII with simple `GET` examples.
5. **Rane HAL** — TCP 4999; example GET/SET strings + parsing.
6. **Shure (REST/JSON)** — GET state with **json_path** (e.g., `channels.1.gain`) and PATCH/PUT for control.
7. **Powersoft (REST/JSON)** — similar JSON approach (gain in dB, mute boolean).
8. **Crestron/AMX HTTP gateway** — unchanged (acts as a generic WS gateway; no polling).

How to use

1. Pick a folder (e.g., `tesira_ttp_tw`), copy `config.example.json` to `config.json`.
2. Edit **IP/port**, `command_template`, and `feedback` fields to your exact objects/IDs.
3. Build & run:

```
go build -o bridge main.go
```

4.
```
./bridge -config ./config.json
```
5.

6. Watch dMix GUI values follow device changes via the feedback pollers.

Tips

- Use `scale` blocks to convert dMix 0..1 ➜ device units (dB, %).
- Tighten `interval_ms` carefully (500–2000 ms is typical).
- If your device needs authentication, add headers or login steps in the REST/ASCII code (simple to extend).

We are happy to help any integrator with further instructions about connecting dMix128 to their infrastructure.