Iziko Constantia AV — Technical Handover

2026-06-02

Constantia BrightSign Toolkit

Internal AV-ops tooling and technical handover for the **Iziko Constantia** immersive AV install: a 6-player synchronised video installation across 5 gallery rooms, with audio served by a Q-SYS Core and Epson projectors controlled by Q-SYS. The show starts and stops on a daily schedule, fully automated, **no operator needed** day-to-day.

> **Private repo. Client deliverable. Not for redistribution.**
>
> Audience: the technical team taking over the install. Snapshot date: **2026-06-02**.

> 🔧 **On site? Read § Field troubleshooting runbook first.** The golden rule on site: **you MONITOR and RECOVER — you do NOT PATCH.** Every near-disaster in the field came from editing the *running* system. Content and firmware changes are done at the **bench, then swapped in** (see ON-SITE RULES).

Table of contents

1. Current state
2. What the system is
3. Network map
4. Architecture — no-reboot, Core-centric
5. Software components and versions
6. The two sync systems
7. BrightAuthor:Connected presentation design
8. Q-SYS integration
    - Talking to the Core (QRC)
    - Component inventory
    - Audio player control pins (verified)
    - Fire procedures (verified)
    - Orderly show shutdown
    - Master Text Controller wiring
    - Master Lua knobs
9. Daily operation
10. On-site PC (the "Wall" / .48)
11. roSync and audio-sync setup (build from scratch)
12. Deploy and re-inject procedures

13. Monitoring
14. Repository layout
15. Quick start (build and install)
16. In-app updates and dashboard
17. Security notes
18. Operational gotchas
19. Field troubleshooting runbook
20. Versions and changelog
21. Local-only dependencies (not in repo)
22. Related documents

Current state (2026-06-02)

Component	Version	Status
Wall app (`bs-provisioner/`)	**wall-v0.11.24**	Working — status grid, S0 stopwatch, diagnostics flush, SyncReceiver Lua in the Q-SYS Setup tab
Master Lua (`qsys-bs-bridge/bridge_controller_master.lua`)	**v2.21.2**	Lean — RebootPulse + player UDP Send pins (Start/Stop/Pause/Resume); audio/markers/stopwatch moved to `sync_receiver`; F5 inert; PREFLIGHT/WAIT_FOR_READY off
SyncReceiver Lua (`qsys-bs-bridge/sync_receiver.lua`)	**v1.7.0**	Separate Text Controller: owns :7000, S0/L0 → StartTrigger + Enable gate, per-player sync monitor
Player Lua (`qsys-bs-bridge/bridge_controller_player.lua`)	**v2.3.1**	Guard 30 s + console logging; reboots only via `Reboot` input ← master RebootPulse
roSync video lock	all players on firmware **9.1.93.2**, equal fps + frame count	Working (firmware parity was the blocker)
Audio fan-out 16/16	verified controls + boot-cached handles	Working — `CONFIRMED playing 16/16`
Auto-start schedule	baked default **09:55–16:45 all days** (`SCHED_DEFAULT`); web schedule overrides	Working

Open items (parked)

1. ✅ `PLAYER_UDP_PORT` — confirmed `5000` (master Send) = BA UDP receiver port `5000`.
2. ✅ **Master video frame-count** — Master now loads `MasterUDP.mp4` (h264 / 25 fps / **20402 frames** / 816.08 s), matching the fleet. All 6 videos equal frame count.
3. ✅ **Window2 PTP domain** — was `0` (own sync group → intermittent drift); now `6`, matching the fleet.

4. **Bench-test** the no-reboot state machine (Leader + 1 Follower) before all 6 — confirm roSync re-locks on `Start` and the sync monitor reads `LOCKED 6/6`.
5. **`locate` seek precision** — confirm `Audio_Player.progress` format (numeric `Value` in seconds vs string-only time readout); decides whether `SyncDrift` can auto-compute Δ.
6. **Add the `SYNC:<id>` heartbeat events** to all 6 `.bpfx` (one per player) so the v1.7.0 sync monitor (`SyncLocked` / `SyncReport`) has data to score.

What the system is

A 6-player synchronised video installation across 5 gallery rooms. The BrightSign players are **video only** — the gallery **audio is served by the Q-SYS Core** (16 audio players → bus → amp). Projectors are Epson, controlled by Q-SYS. The show runs on a daily schedule with no operator required.

The architecture is **no-reboot and Core-centric** (pivot of 2026-06-02): each player runs a BrightAuthor:Connected state machine, the Core drives and receives everything **directly**, and the on-site laptop is no longer in the live signal path — it can sleep or fail and the show still runs.

Network map

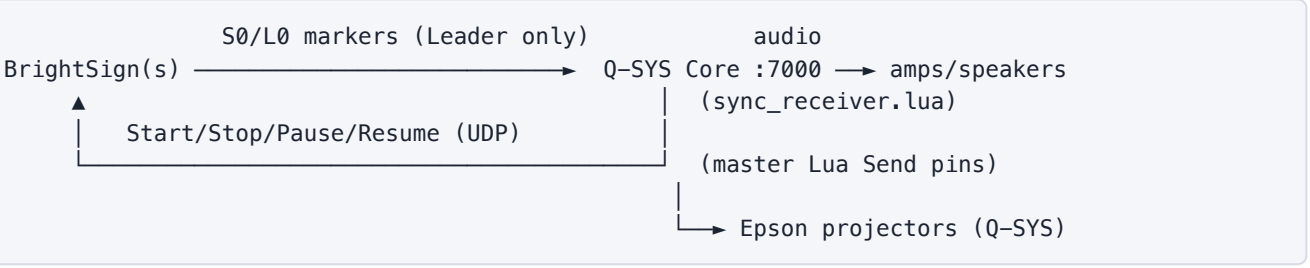
Isolated static LAN — **no gateway, no router, no internet, by design.** All traffic is on-subnet (`192.168.1.x ↔ 192.168.1.x`).

Device	IP	Model	Role
Q-SYS Core	`192.168.1.10`	—	Gallery audio (16 audio players → bus → amp), schedule, show control. Listens UDP **:7000** for S0/L0 markers. QRC (JSON-RPC) on **:1710**.
Master BrightSign (Leader)	`192.168.1.11`	**LS445**	roSync Leader. Emits S0/L0 markers. Video only (no audio out). MAC `90:ac:3f:32:36:a8`, switch port `1/0/11`.
Office 1 player	`192.168.1.30`	LS425	Follower
Player	`192.168.1.31`	LS425	Follower
Player	`192.168.1.32`	LS425	Follower
Player	`192.168.1.33`	LS425	Follower
Player	`192.168.1.34`	LS425	Follower
On-site PC ("Wall")	`192.168.1.48`	Windows	`IzikoQSYSLaptop`. Monitoring + authoring + remote access + embedded bridge (:5000). **Out of the live signal path** — see Architecture.
Managed switch	`192.168.1.2`	TP-Link TL-SG3452XP	Flat switch. (Historically a stray `gw=192.168.1.2` was mis-written onto the Master — see T-1.)
Example projector	`192.168.1.40`	Epson	One of 5 Epson projectors controlled by Q-SYS.

- Subnet: `192.168.1.0/26` (mask `255.255.255.192`), flat, static, no DHCP/router.

Architecture — no-reboot, Core-centric

The old model rebooted the whole fleet daily to force a frame-0 sync. The new model (2026-06-02 pivot) replaces that with a **BrightAuthor:Connected state machine** on each player plus **Core-direct UDP** in both directions.



- **Markers in:** the Master BrightSign sends S0/L0 to **Core** `192.168.1.10:7000` **DIRECT.** The BA global UDP destination on **every SD card** is the Core, **not** the Wall.
- **Control out:** the Q-SYS Core sends Start/Stop/Pause/Resume to the players **direct** via the master Lua UDP Send pins (`PLAYER_IPS` = `.11` + `.30–.34`, on `PLAYER_UDP_PORT` default `5000`).
- **Audio and projectors** are Core-local.
- **The `.48` laptop is out of the live signal path entirely** — markers go straight to the Core, control comes straight from the Core, audio and projectors are on the Core. The laptop is authoring and monitoring only and can sleep or fail without stopping the show.

roSync re-locks on each Start, so **no reboot is needed** day to day. A reboot (per-player `RebootPulse`, or the optional morning PC reboot) remains only as a manual recovery fallback for the rare case roSync loses lock.

Why no daily reboot anymore

roSync (BrightSign Player Sync / PTP) frame-locks the fleet **while playing** and **re-converges on each Start** — it does not need a reboot to lock. Pushing the players to the `Show` state with a Synchronize on each Start re-anchors all six at frame 0; the Leader's `S0`@0 then seeks the Core audio to 0. This removes the fragile reboot timing entirely.

Software components and versions

Component	Version	Role
`qsys-bs-bridge/bridge_controller_master.lua`	**v2.21.2**	Core "master" Text Controller. Daily schedule, show flip-flop drive, projector watchers, web command poll, **RebootPulse** (per-player reboot, manual recovery only), **Send pins** (Start/Stop/Pause/Resume UDP → players). Audio/markers/stopwatch were moved to `sync_receiver`.
`qsys-bs-bridge/sync_receiver.lua`	**v1.7.0**	Separate Text Controller. Owns **:7000**; S0/L0 → `StartTrigger` + `Enable` gate; forwards markers to the Wall :7000 (display-only, loop-safe). Per-player sync monitor parses `SYNC:<id>` heartbeats → `SyncLocked` / `SyncSpreadMs` / `SyncReport`.
`qsys-bs-bridge/bridge_controller_player.lua`	**v2.3.1**	Per-player Text Controller. Reboots its player only on the `Reboot` input (← master `RebootPulse`). Startup guard 30 s + console logging. Manual single-player control only — keep it off the global Play net.
Wall app (`bs-provisioner/`)	**wall-v0.11.24**	Electron monitoring/provisioning app + embedded bridge (:5000). Q-SYS Setup wizard copies all 3 Lua files. Also listens UDP **:7000** for the Core's forwarded marker feed (display-only, no relay).
BrightAuthor:Connected presentations	per-room `.bpfx`, archived in `brightsign-presentations/`	The player state machine (see presentation design).

> **Two copies of the master Lua exist** — the standalone `qsys-bs-bridge/bridge_controller_master.lua` and the wizard template `LUA_MASTER_TEMPLATE` in the Wall renderer. Update both. The standalone uses the helper `fireOutputTrigger`; the wizard uses `fireOut` — do not cross-paste snippets between them.

The two sync systems

Do not conflate these — they are independent layers.

Layer	Syncs	Mechanism	Where configured
roSync (Player Sync, "Video↔Video")	the 6 videos to each other	BrightSign `roSyncManager` PTP, peer-to-peer on the LAN	BA → Networking → Player Sync (Leader/Follower + Domain 6)
Audio-sync ("Audio↔Video", S0/L0)	Q-SYS audio to the Master's video frame-0	Master sends UDP markers → Core `sync_receiver` re-seeks the audio	BA UDP events (Master only) + Q-SYS Lua

Separate rooms with separate projectors still need both: **roSync** so all rooms run at the same playback position, and **audio-sync** so the Q-SYS gallery audio tracks the Master. (roSync frame-lock matters most for a tiled video wall; here each room loops its own video, but the fleet is still kept frame-locked so the single audio bed stays aligned to the Master.)

BrightAuthor:Connected presentation design

Each player runs one BrightAuthor:Connected presentation, authored as a `.bpfx` project. All 6 source projects are archived under `brightsign-presentations/` (one folder per player) so any one can be rebuilt if a source machine is lost. `MasterGCUDP` is the Sync Leader that emits the S0/L0 markers; `Office1/2`, `Window1/2`, `DrawingRoom` are Followers. The published `pool/` media (video assets, >100 MB) is **not** committed; video masters live in `…/GrootConstantia/VIDEOs/`.

Two states per player (authored identically across all 6)

State	Content	Player Sync	Notes
`Black` / park (INITIAL state)	a per-room ID placard image (`brightsign-presentations/park-images/`)	n/a	Boots **silent, no video, no markers**. Setting this as the initial state makes the *player* side inert on boot — a reboot/F5/power-up no longer auto-plays or spams markers; the player waits in black until a real Start.
`Show`	the room's looping clip (`Loop` ON)	**Leader** (Master) / **Follower** (rest), same Sync Domain	The running show. roSync locks here and re-converges on each Start.

Events

- **UDP `Start`** → `Show` (the clip restarts at **frame 0**; fire a **Synchronize** so the fleet enters `Show` together and roSync locks at frame 0).
- **UDP `Stop`** → `Black` (video stops → markers stop → screens park).
- **UDP `Pause` / `Resume`** → video-only freeze/continue. Momentary only — **not** for the daily start. `Resume` continues from the frozen frame, **not** frame 0; use `Start` for a clean frame-0 anchor.
- **Markers (LEADER only): Video Timecode @ 0 ms → UDP `S0`** to Core `:7000`. Optionally add more Video Timecode senders at 60000 / 120000 / 180000 ms (all sending `S0`) so audio re-syncs every minute and drift is bounded to <1 min instead of one re-sync per 13.6-min loop. `L0` (Media-End) is an optional/backup loop-boundary marker. Use **Video** Timecode, not Audio Timecode — the clips are video-only.

- **Sync heartbeat (ALL players, for the v1.7.0 monitor):** Video Timecode event(s) at a fixed loop position (e.g. every 60–120 s) → UDP `SYNC:<id>` to Core `:7000`, where `id` is `M` / `DR` / `O1` / `O2` / `W1` / `W2`. This lets the Core compute fleet-lock health (arrival spread). It is a **distinct keyword from `S0` / `L0`** and does not touch audio.

> **Key facts the state-machine design rests on:**
>
> - BrightSign "Video Timecode" = media **playback position**, not a wall clock. A Video Timecode event @ 0 ms fires when the clip reaches frame 0 on each loop — it is a *detector* ("video is at 0 now"), it does NOT command the video to 0.
> - **Pause → Resume continues from the frozen frame, not 0.** To return all players to frame 0 you must **restart the state** (re-enter `Show`), not resume.
> - **A player can't power itself off** (always-on). "Stop" = go to the `Black` state, not power-off.

Park placards (in `PARK_IMAGES`, generated)

- **Office1** player is **PORTRAIT** (1080×1920) → `PARK_01_1080x1920.png` ("O1").
- **Office2** player is **LANDSCAPE** (1920×1080) → `PARK_02_1920x1080.png` ("O2").
- Also `PARK_DR`, `PARK_M`, `PARK_W1`, `PARK_W2` (all 1920×1080).

Videos

In `VIDEOs/UpdatedOptimizedMay2026/Update/b/`: all 6 clips are **h264, 25 fps, 20402 frames, 816.08 s (13:36)**. Five are 1920×1080 landscape; Office1 is portrait 1080×1920. The Master loads `MasterUDP.mp4`. roSync requires the **same fps + same frame count** (met); resolution may differ. PTP / Sync Domain = **6** on all 6; Leader = the Master only.

Q-SYS integration

Talking to the Core (QRC)

QRC API: **`192.168.1.10` : `1710`**, JSON-RPC 2.0, messages **NUL-terminated** (`\0`). Responses too. Design name: `Groot Constansia Museum` (Q-SYS Designer 10.1.0 — note the misspelling "Constansia").

macOS / Linux (`nc`):

```
printf '{"jsonrpc":"2.0","method":"Component.GetComponents","params":"","id":1}\0' | nc
        192.168.1.10 1710
```

Windows (Python — `nc` absent):

```
import socket
s = socket.create_connection(("192.168.1.10", 1710), timeout=4)
s.sendall(b'{"jsonrpc":"2.0","method":"Component.GetComponents","params":"","id":1}\x00')
buf=b""; s.settimeout(1.5)
try:
    while True:
        c=s.recv(32768)
        if not c: break
        buf+=c
        if buf.count(b"\x00")>=1: break
except socket.timeout: pass
print(buf.split(b"\x00")[0].decode())
```

> **QRC visibility:** `Component.GetComponents` only returns components whose **Script Access = External or All**. Components set to `Script` or `None` exist but are invisible to QRC (e.g. the master Lua's Text Controller). QRC External Control must stay enabled on the Core.

Component inventory (running design)

Audio players — 16 × `audio_file_player`. Code Names are **non-contiguous**: there is NO `Audio_Player_8`, but there IS `Audio_Player_16`.

```
Audio_Player, Audio_Player_1..7, Audio_Player_9..16   (16 total: base + _1..7, _9..16)
```

Zone map (from each player's `filename` control):

Code Name	Zone	Code Name	Zone
`Audio_Player`	O1	`Audio_Player_7`	H3
`Audio_Player_1`	O2	`Audio_Player_9`	D1
`Audio_Player_2`	O3	`Audio_Player_10`	D2
`Audio_Player_3`	O5	`Audio_Player_11`	D3
`Audio_Player_4`	O4	`Audio_Player_12`	D4
`Audio_Player_5`	H1	`Audio_Player_13`	D5
`Audio_Player_6`	H2	`Audio_Player_14`	D6
		`Audio_Player_15`	D7
		`Audio_Player_16`	D8

Projectors — 5 × Epson plugin (contiguous): `EpsonProjector`, `EpsonProjector_1` … `EpsonProjector_4`, watched on `power.state`.

Show-state latches — `flip_flop`:

- **`Flip-Flop`** — show ON/OFF latch. `state=true` ⇒ show running.
- **`Flip-Flop_Shutdown`** — drives the orderly shutdown routine (see below).

Audio player control pins (verified)

⚠️ These are the **real** controls, verified live against the running Core on 2026-06-01 via QRC. The names assumed earlier in development (`play.state`, `transport.play`, `playback.position`) **do not exist** on this component type — that mistake caused a long "no-component" hunt.

Control	Dir	Use
`play`	trigger	**start playback** (from current position)
`stop`	trigger	**stop playback**
`pause`	trigger	pause
`locate`	write	**seek to position (seconds); Value `0` = top ("rewind")**
`loop`	bool	loop track
`gain` / `mute`	R/W	level / mute
`filename`, `directory`, `root`	R/W	track selection
`playing`	RO bool	**`true` while playing — verify here**
`stopped`	RO bool	`true` while stopped
`progress` / `remaining`	RO	`hh:mm:ss`
`status`	RO	e.g. `OK: 48kHz, 24-bit, stereo`

> `Component.New` is reliable at script **init only** — it returns nil inside Timer/UdpSocket/HTTP callbacks. Resolve and **cache handles at boot**.

Fire procedures (verified)

Audio ON (play from top) — per player. Send all 16 `locate` first, then all 16 `play`, so they start together:

```
{"jsonrpc":"2.0","method":"Component.Set","params":{"Name":"Audio_Player","Controls":
        [{"Name":"locate","Value":0}]},"id":1}
{"jsonrpc":"2.0","method":"Component.Set","params":{"Name":"Audio_Player","Controls":
        [{"Name":"play","Value":1}]},"id":2}
```

Audio OFF — per player:

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

Verify:

```
{"jsonrpc":"2.0","method":"Component.Get","params":{"Name":"Audio_Player","Controls":
        [{"Name":"playing"},{"Name":"stopped"},{"Name":"progress"}]},"id":1}
```

In-design Lua (what the master/`sync_receiver` Lua does):

```lua
for _, name in ipairs(AUDIO_PLAYERS) do
  local p = Component.New(name)        -- cached at BOOT; nil inside async callbacks
  if p and p["play"] then
    p["locate"].Value = 0              -- rewind to start (or to S0 offset)
    p["play"]:Trigger()                -- play
  end
end
-- Stop: if p and p["stop"] then p["stop"]:Trigger() end
```

Quick crib (macOS `nc`):

```
CORE=192.168.1.10; P=1710
printf '{"jsonrpc":"2.0","method":"Component.GetComponents","params":"","id":1}\0' | nc $CORE
        $P
printf '{"jsonrpc":"2.0","method":"Component.GetControls","params":
        {"Name":"Audio_Player"},"id":1}\0' | nc $CORE $P
# play O1 from top
printf '{"jsonrpc":"2.0","method":"Component.Set","params":{"Name":"Audio_Player","Controls":
        [{"Name":"locate","Value":0}]},"id":1}\0' | nc $CORE $P
printf '{"jsonrpc":"2.0","method":"Component.Set","params":{"Name":"Audio_Player","Controls":
        [{"Name":"play","Value":1}]},"id":1}\0' | nc $CORE $P
# stop O1
printf '{"jsonrpc":"2.0","method":"Component.Set","params":{"Name":"Audio_Player","Controls":
        [{"Name":"stop","Value":1}]},"id":1}\0' | nc $CORE $P
```

> **Safety:** `play` and projector `PowerOn` produce **real** sound and light in the museum. Confirm scope (one zone vs all 16) before firing broadly.

Orderly show shutdown

The master Lua's shutdown routine fires when `Flip-Flop_Shutdown` gets a clean falling edge. A lone reset does **not** fire it (no edge if already false). Pulse **set → wait ~1 s → reset**:

```
{"jsonrpc":"2.0","method":"Component.Set","params":{"Name":"Flip-Flop_Shutdown","Controls":
        [{"Name":"set","Value":1}]},"id":1}
// wait ~1s
{"jsonrpc":"2.0","method":"Component.Set","params":{"Name":"Flip-Flop_Shutdown","Controls":
        [{"Name":"reset","Value":1}]},"id":2}
```

Verified result: `Flip-Flop` (show) → `state=false`; all 5 projectors → `PowerStatus = "Standby Mode (Network ON)"` (laser off, network reachable); audio `0/16` playing. The master Lua does this automatically on schedule-OFF and web Stop via `driveShutdown()` / `showOff()`.

Projector controls (`EpsonProjector*`):

Control	Use
`PowerOff`	trigger → Standby (Network ON)
`PowerOn`	trigger → power on
`Power` (RO bool) / `PowerStatus` (RO)	`"Laser ON"` vs `"Standby Mode (Network ON)"`
`AVMute`	blank without powering down

Master Text Controller wiring

Pin	Dir	Wire to
`StartPlay` / `StopPlay`	in (Trigger)	Play / Stop buttons
`WebToggleSet` / `WebToggleReset`	out (Trigger)	show flip-flop `Set` (ON) / `Reset` (OFF)
`RebootPulse`	out (Trigger)	each per-player Lua `Reboot` input (manual recovery)
`SendStart` / `SendStop` / `SendPause` / `SendResume`	in (Trigger)	show flip-flop ON → `SendStart`, OFF → `SendStop`; buttons → Pause/Resume
`Status`	out (Text)	optional UCI readout

`sync_receiver` pins: `StartTrigger` (Trigger out → delayed-start), `Enable` (Bool in ← show flip-flop), `Status` / `Log` / `SyncMarker` (Text out → monitor), plus the v1.7.0 monitor pins `SyncLocked` (Bool), `SyncSpreadMs` (Str), `SyncReport` (Str).

Canonical flip-flop scheme: two master output Trigger pins — `WebToggleSet` → flip-flop `Set` (ON) and `WebToggleReset` → flip-flop `Reset` (OFF). These mirror the manual Set/Reset controls, which clear the latch both ways. **Never write** `State` (it is a read-only output). If the pins are not wired, the Lua falls back to `Component.New("Flip-Flop")` — which needs the component's **Script Access = `Script`** and the correct Code Name.

Config to confirm in the master Lua: `PLAYER_UDP_PORT` (must match BA's UDP-receive port), `PLAYER_IPS` (Master `.11` + `.30–.34`), and `SEND_KEYWORDS` (must match BA's "Specify UDP input").

Master Lua knobs

Top of `bridge_controller_master.lua`:

Knob	Default	Purpose
`SCHED_DEFAULT`	`09:55–16:45, all days, enabled`	**Auto-start baked default.** Runs with no web schedule; a web-set schedule (valid times) overrides; a *blank* web schedule does NOT clobber it.
`SCHEDULE_ENABLE`	`true`	Run the on-Core schedule executor.
`STARTUP_GUARD_SEC`	`30`	Blocks runPlay/runStop REBOOT for the first 30 s after F5/boot (Q-SYS re-asserts latched pins on load).
`DRIVE_AUDIO_DIRECT`	`true`	Drive Audio Players via `Component.New` (vs wired pins).
`USE_SHUTDOWN_ROUTINE`	`true`	OFF uses the `Flip-Flop_Shutdown` set→reset edge (projectors → Standby).
`WATCHERS`	5 Epsons	Components polled for state changes (projectors), on `power.state`.
`DEBUG_AUDIO`	`true`	Log per-bus seek path. Turn off once confirmed.
`AUDIO_POLL_ENABLE`	`true`	10 s "N/total playing" readback poll.

> `sync_receiver` **owns the audio re-sync** in the current split. Historically the master Lua held S0/L0 handling under these knobs: `SYNC_LOOP_ENABLE` (master switch — must be `false` **on the master** now, since only one component may bind :7000), `SYNC_LOOP_PORT` (`7000`), `SYNC_START_OFFSET_SEC` (must equal the BA Timeout "Time on screen", default `0.1`), `SYNC_LOOP_MIN_GAP_SEC` (debounce, `5`), and the in-Core stopwatch trio `SyncStopwatch` / `SyncAudioPos` / `SyncDrift`.

Optional Text controls to surface on the schematic/UCI (Script Access = Script): `SyncLastMarker` (last S0/L0 + counts), `AudioPlayState` (N/total playing), and the sync-stopwatch trio `SyncStopwatch` / `SyncAudioPos` / `SyncDrift` (the video-loop clock, the reference audio `progress`, and their difference; no-op if the pins are absent).

Daily operation

```
(idle: all 6 players in Black/park, silent — roSync idle)
09:55  Core schedule edge → flip-flop ON  → SendStart (all 6) + projectors ON + audio Enable
        → players Start → Show @ 0, roSync locks, Leader S0@0 → Core seeks audio to 0 + plays
16:45  Core schedule edge → flip-flop OFF → SendStop (all 6) + projectors Standby + audio stop
+ Enable off
        → players → Black/park
```

- The baked schedule (**09:55–16:45 all days**, master Lua `SCHED_DEFAULT`) runs even with no web schedule set. A web-set schedule (valid times, via the dashboard) overrides it; a *blank* web schedule does NOT clobber the baked default.
- **F5/boot is inert** — loading the design to the Core does NOT reboot players, turn on projectors, or start audio. It runs the design and sits idle (audio gate CLOSED, schedule seeded OFF). Only a real schedule edge (or web Play) starts the show. See Operational gotchas for why.
- The morning reboot is now **optional** (a roSync-recovery fallback). If you keep it, see On-site PC for the reboot → auto-login → shell:startup F5 mechanism and why a timed F5 task does not work on its own.

On-site PC (the "Wall" / .48)

- **Host** `IzikoQSYSLaptop` · `192.168.1.48` · Windows user `Iziko` · remote via Google Remote Desktop (remotedesktop.google.com).
- **Auto-login is ON** — Winlogon `AutoAdminLogon=1`. The PC boots straight to the `Iziko` desktop so the shell:startup Q-SYS autostart can run unattended. Verify:

  ```
  reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon" /v AutoAdminLogon
  :: expect 1
  ```

- **Daily reboot scheduler** — `tools/schedule-qsys-reboot.ps1` registers the task "Constantia Morning Reboot": reboot → auto-login → shell:startup `constantia_qsys_autostart.exe` fires Q-SYS Designer F5 (Save to Core & Run) so the design loads in the interactive session. This was the **reboot-model** daily load; under the no-reboot pivot it is **optional** (fallback only).

```
powershell -ExecutionPolicy Bypass -File tools\schedule-qsys-reboot.ps1 -RebootTime
        "09:40"
Get-ScheduledTask -TaskName "Constantia Morning Reboot" | Format-List TaskName,State
```

Requires `AutoAdminLogon=1` **and** `constantia_qsys_autostart.exe` present in shell:startup. (`schedule-qsys-load.ps1` is **deprecated** — an F5 driven from a Task Scheduler task does not land, because the task can't take foreground input on the interactive desktop.)

- 🔒 **The Windows login password is NOT in this repo** (it is a credential / client deliverable). It lives in **the gated dashboard (Ops → Remote access) + the operator's secure note**. Rotate it in one place; do not paste it into git.

---

roSync and audio-sync setup (build from scratch)

Bench-build and test, then deploy by SD swap — never patch a live player (see ON-SITE RULES). Don't author from scratch unless you have to: open the relevant `.bpfx` from `brightsign-presentations/`, re-link the video asset, and re-publish. This section is the from-scratch reference and the checklist to verify an archived project still has every mandatory setting.

Part A — roSync (video ↔ video frame-lock)

Prerequisites (ALL mandatory — miss one and it silently won't lock):

1. **Same firmware on all 6 players.** Mixed 9.0.x / 9.1.x silently breaks PTP. Target **9.1.93.2** on every player. Verify per player: `curl -s "http://<ip>/api/v1/info"` → check `FWVersion`. (Model may differ — LS445 vs LS425 mixing is fine; only the OS version must match.) OTA updates can silently fail (the file ends up `cobra-update.bsfw_invalid` on the SD, the player stays on the old OS). If OTA won't take, flash via SD recovery (fresh SD with only the `.bsfw`).
2. **Identical fps + identical FRAME COUNT per clip.** roSync frame-locks; a clip with more frames falls behind by the difference EVERY loop and drifts forever. Constantia value: **25 fps CFR, 20402 frames** (= 816.08 s) on ALL clips. Resolution may differ per screen (landscape 1920×1080 vs the portrait rotated 1080×1920) — resolution does NOT affect sync; fps + frame count do. Verify (count frames, do not trust duration):

   ```
   ffprobe -v error -select_streams v:0 -count_frames \
     -show_entries stream=width,height,r_frame_rate,nb_read_frames \
     -of default=nw=1 "clip.mp4"
   # All clips MUST report the SAME nb_read_frames and the SAME r_frame_rate.
   ```

   Re-encode to the common spec with `tools/encode-videos.sh` (enforces TARGET_FRAMES + same fps + CFR + stripped audio + faststart). To trim a single long clip:

   ```
   ffmpeg -y -i "Office1_RotatedVideo.mp4" -frames:v 20402 \
     -c:v libx264 -profile:v high -pix_fmt yuv420p \
     -b:v 2M -maxrate 2M -bufsize 4M -g 25 -keyint_min 25 -sc_threshold 0 \
     -r 25 -vsync cfr -an -movflags +faststart "Office1_20402.mp4"
   ```

3. **Wired Ethernet, same subnet, multicast allowed** (a flat switch is fine).

BA:Connected — Player Sync settings (per presentation), Presentation → Edit → Interactive → Networking → Player Sync:

Setting	Master (.11)	Followers (.30–.34)
Enable	✅ checked	✅ checked
Role	**Leader**	**Follower**
Domain	**6**	**6** (same on all)

- The Domain number IS the sync group — all 6 must share it (we use **6**).
- Exactly **one Leader.** Two leaders on the same domain → followers lock to the wrong one and the real Leader drifts alone.
- Each player's timeline = its video state set to loop: **Media End event → Remain on current state → Continuous.** No Synchronize command is needed for a single looping state when all clips share length + fps + domain. (Synchronize is for coordinating multi-state transitions, e.g. the Start re-anchor in the no-reboot model.)

Verify roSync — trust the screens, not net-doctor. On fw 9.x, net-doctor's "roSyncManager Leaders: none announced" can be a **false negative** (it sniffs the legacy multicast; fw 9 uses PTP on a different group). Look at adjacent displays: same playback position over a full loop = locked. If one player drifts while the rest lock, check that player's **firmware** and **frame count** first (the two real causes).

Part B — audio-sync (Q-SYS audio ↔ Master video, S0/L0 markers)

The Master sends UDP markers; the Core's `sync_receiver` Lua re-seeks the gallery audio to match. Under the no-reboot pivot the global UDP destination on every SD card is the **Core** `192.168.1.10:7000` **DIRECT** (the historical Wall-relay path, Master → Wall :5000 → Core :7000, is now dormant — see Architecture).

BA:Connected — author the markers (MASTER ONLY). On the video (`Show`) state:

- **Event 1 —** `S0` **(start / frame-0):** Event type **Video Timecode @ 0 ms** (the no-reboot design) — or, historically, a **Timeout** event with **Time on screen = 0.10 s** (BA has no true "media start" event; Timeout 0.10 fires just after frame 0, and the Lua's `SYNC_START_OFFSET_SEC` = 0.10 compensated — keep them equal). Command **Send → UDP**, message/data field = literal `S0` (2 bytes). Target State: Remain on current state.
- **Event 2 —** `L0` **(each loop boundary):** Event type **Media End**. Command **Send → UDP**, message = literal `L0` (2 bytes). Target State: **Remain on current state → Continuous** (loops the video).

CRITICAL payload notes (these bit us in the field):

- The Send-UDP **message/data** field must contain the literal `S0` / `L0` — **NOT** the IP address. (Typing the IP sends the IP as the payload; the Lua matches on `S0` / `L0`, so it never fires. The old bug showed as 17-byte packets with payload `192.168.1.10:7000`.)
- Use plain **Send UDP.** Do NOT use "Send UDP **bytes** (comma-separated)" — that emits a 1-byte blank, not the token.
- A correct marker appears in the Wall's 📡 UDP Listener as a **2-byte** packet from `192.168.1.11` with payload `S0` / `L0`.
- **Followers send NOTHING** — only the Master (Leader) emits S0/L0.

Q-SYS Core side:

- The 16 Audio Players must have Code Name `Audio_Player`, `Audio_Player_1` … `Audio_Player_15` (+ `_16`, no `_8`) and **Script Access = Script** (else `Component.New` returns nil → log shows `no-component`).
- After editing the design, **F5 (Save to Core & Run)** to push it (an unsaved `*` in the title bar = the running Core doesn't have your changes yet).
- The `sync_receiver` boot log should report it is listening on UDP 7000.

Verify audio-sync — test without waiting the full 13:36 loop:

```
node tools/sync-fire.js --profile constantia --loops 3 --gap 6   # fires S0 + 3× L0 at the Core
node tools/sync-fire.js --token L0          # one L0 and exit
node tools/sync-fire.js --token S0          # one S0 and exit
```

Watch the Core Event Log (`recv L0 … → re-sync audio`, `audio start: … CONFIRMED playing N/16`) and, if still on the relay path, the Wall's 🔁 relay banner (`S0:n · L0:n` climbing).

Part C — deploy (bench + swap, never live-inject)

1. **Bench-build the SD:** in BA:Connected, open the player's presentation → confirm the video (correct frame count) + Player Sync + (Master only) the S0/L0 events + the `Black`-initial state + the global UDP dest = Core `192.168.1.10:7000` → **Publish** → **Standalone** to a fresh-formatted SD via a card reader. (Do NOT publish "Setup files only" — that gives the "Congratulations… set up!" splash and never plays.)
2. **Strip Mac junk** before ejecting (or it may not boot clean):

   ```
   SD=/Volumes/<SD_NAME>
   dot_clean "$SD"
   rm -rf "$SD/.Spotlight-V100" "$SD/.fseventsd" "$SD/.Trashes" "$SD/.DS_Store"
   ```

3. **Test on the bench** — does it play and (with peers) lock?
4. **Swap** the SD/unit in on site, power on. If bad → swap back. Reversible.

> **The video lives in `pool/` by sha1, mapped by `local-sync.json`.** A bare `.mp4` upload does nothing — the player looks up the sha1 in the manifest, not the filename. A new/trimmed clip = new sha1, so `local-sync.json` AND the new `pool/` blob must update together. **Re-publish in BA (Standalone)** so it regenerates the manifest with the new sha1 + frame-count probe, then push the whole bundle with the Wall's 🔧 **Deploy BA bundle**. Do NOT hand-edit `local-sync.json` or upload the blob alone. (Skip re-uploading `cobra-update.bsfw`, ~447 MB, already applied.) Verify: `curl -s "http://<ip>/api/v1/files/sd/local-sync.json"` and confirm the clip's `probe` shows the new frame count.

Bench test before trusting the no-reboot state machine (Leader + 1 Follower)

1. Both on fw 9.1.93.2, same clip fps + frame count, Player Sync Leader/Follower on the same Domain, both INITIAL state = `Black`.

2. Send `Start` to both → both restart `Show` @ 0, roSync locks (watch for a clean frame-0 start), Leader emits `S0` → Core audio @ 0.
3. Let it loop a few times → confirm S0 fires each loop (or each minute if multi-point) and audio stays aligned.
4. Send `Stop` → both go `Black`, markers stop.
5. Power-cycle one → it boots to `Black` (no auto-play, no markers) → `Start` brings it back in sync.

If all hold, roll to all 6 and retire the daily reboot.

> **Caveats to confirm on the bench:** on each Start the followers re-converge to frame 0 and may show a brief (sub-second) frame jump — acceptable for a gallery? Verify BA's Synchronize actually coordinates the state-change across the sync group on your firmware. The INITIAL state MUST be `Black` in every bpfx or that player auto-plays + spams markers on boot.

The values that must match across all 6:

Invariant	Value
Firmware	**9.1.93.2**
FPS	**25 CFR**
Frame count per clip	**20402** (816.08 s)
Player Sync Domain	**6**
Leaders	exactly **1** (the Master)
Audio markers	Master only: `S0` + `L0`, payload `S0` / `L0` to Core `:7000`

Deploy and re-inject procedures

- **Q-SYS Lua:** open each Text Controller in Designer, paste the matching file from `qsys-bs-bridge/`, **F5** (Save to Core & Run). There are three controllers: **master** + **sync_receiver** (only `sync_receiver` binds :7000) + **6× player**.
- **Wall app:** run `tools/update-and-build.bat` on the on-site PC (`git pull` + `npm install` + `sync:shared` + electron-builder NSIS, then opens `dist\`), or push a `wall-v*` tag for CI. Auto-update feed = Cloudflare R2/Pages (studiouih account).
- **Dashboard:**

```
cd update-feed && npx wrangler@3 pages deploy site --project-name constantia-updates --
        branch main --commit-dirty=true
```

- **SD cards:** author the 6 `.bpfx` (states + park image + dest = Core `:7000` + Leader/Follower), publish **Standalone**, deploy via the Wall (bench + swap).

Monitoring

- **On the Core (no laptop):** the `sync_receiver` `Status` / `Log` / `SyncMarker` pins (marker counts, last-marker age, drift). **Fleet-lock health (v1.7.0):** `SyncLocked` / `SyncSpreadMs` / `SyncReport`, driven by the per-player `SYNC:<id>` heartbeats. `SyncReport` reads e.g. `LOCKED 6/6 spread 12ms` (good), `DRIFT 6/6 460ms — W2 last`, or `INCOMPLETE 5/6 missing: W2` (names the bad player). Resolution is ~tens of ms (UDP jitter + tick) — it catches drift/dropout, **not** sub-frame; for sub-frame use the telnet PTP offset. The master schedule tick logs `WINDOW`, countdown, and `audioGate`.
- **On the Wall:** live screenshots, the Audio/Projector status grid, and the **S0 stopwatch / timeline marks** (revived under Core-direct — the Core's `sync_receiver` forwards markers to the Wall's display-only :7000 listener; needs UDP 7000 inbound, which the installer opens). The Wall's 🔊 panel stopwatch and the in-Core `SyncStopwatch` pin both reset on each S0/L0 — compare to the audio `progress`. The Wall audio-poll's ~10 s lag is display only, not real drift.

Repository layout

Folder	What it is	Built artifact
`bs-provisioner/`	**Constantia BrightSign Wall** — Electron app. Live multi-player screenshot wall (per-tile reboot, fullscreen, network diagnostics, 1 s refresh, fit-to-screen layout), **embedded Q-SYS bridge** on `127.0.0.1:5000`, bulk `.txt` config import, **Q-SYS Setup** wizard (generates the Lua), live request log + connectivity Test tab, **Q-SYS IDs** table, and a **Player Scripts** tab (back up + inject BrightScript over DWS). (Badly-named legacy folder — this IS the Wall.) See `bs-provisioner/README.md` and `bs-provisioner/BUILD-WINDOWS.md`.	`dist/ConstantiaBSWall-Setup-<ver>-x64.exe` (NSIS) + `-Portable-` `.exe` + macOS `.dmg`
`bs-provisioner-classic/`	**Constantia BrightSign Provisioner** — Electron app. Clones a known-good master SD card to N target SD cards while patching per-player config (unitName, network mode, static IP, optional skip-network-diagnostics). Operates *offline* on a mounted card via robocopy + per-player XML patching of `localSetupToStandalone-sync.xml`.	`dist/ConstantiaBSProvisioner-<ver>-x64.exe`
`shared/`	Canonical, dependency-free libraries synced into each Electron project's `src/` at build (`npm run sync:shared`): **`dws-client.js`** (DWS HTTP/UDP client incl. file list/download/upload/delete, HTTP Digest auth) and **`bridge-server.js`** (the embeddable Express bridge — `/api/players` CRUD, broadcast reboot/ready, per-player reboot/snapshot/inject, request log, `snapshot.jpg`, `/qsys-view`).	none
`bs-bridge/`	Thin standalone CLI wrapper around the shared bridge (`createBridge()` factory) + a browser control panel. See `bs-bridge/README.md`.	none
`qsys-bs-bridge/`	Q-SYS-side Lua: `bridge_controller_master.lua`, `sync_receiver.lua`, `bridge_controller_player.lua`, plus `reboot_listener.brs` reference + Node mocks (archive). See `qsys-bs-bridge/README.md`.	release tags `bridge-controller-v<ver>`
`tools/`	Operator + dev helpers: `collect-brs-backups.js` / `inject-brs.js` (round-trip player BrightScript through git), `constantia_qsys_autostart.{ahk,ps1}` (auto-launch Q-SYS Designer on boot — globs `Q-SYS Designer *\Q-Sys Designer.exe` + the newest `Groot Const* Museum v*.qsys`, then **F5**), `schedule-qsys-reboot.ps1`, `encode-videos.sh`, `sync-fire.js`, `net-doctor.js`, `update-and-build.bat`, `gen-docs.js` (auto-CHANGELOG).	autostart bundle tagged `tools-autostart-v<ver>`

Folder	What it is	Built artifact
`update-feed/`	Private-repo-safe cloud backend: `worker/` (the `constantia-dl` Cloudflare Worker — installer feed, OTP auth, telemetry ingest, command + schedule relay) and `site/` (the OTP-gated dashboard PWA on Cloudflare Pages). See `update-feed/README.md`.	Worker + Cloudflare Pages
`brightsign-presentations/`	Source `.bpfx` (BrightAuthor:Connected project files) for all 6 players — one folder per player (`MasterGCUDP` = Sync Leader, the rest Followers) + `park-images/`. The published `pool/` media is NOT here by design; video masters live in `…/GrootConstantia/VIDEOs/`.	none (source archive)

Codebase quick map

```
shared/
  dws-client.js          ← HTTP Digest auth lives here; canonical client
  bridge-server.js       ← createBridge() factory used by Wall + CLI
bs-bridge/
  src/server.js          ← thin CLI wrapper (npm start)
  src/{dws-client,bridge-server}.js  ← synced from ../../shared/
  public/                ← bridge's browser control panel
bs-provisioner/          ← THE WALL
  src/main.js            ← Electron main; embeds bridge; IPC handlers; VERSION_FEED_URL
  src/preload.js         ← contextBridge exposure
  src/renderer/{index.html,renderer.js,styles.css}  ← UI, sync, Q-SYS wizard
  src/{bridge-server,dws-client}.js  ← synced from shared/
  package.json           ← Wall version + build config
bs-provisioner-classic/  ← Provisioner app (SD card cloner)
qsys-bs-bridge/
  bridge_controller_master.lua   ← master Text Controller
  sync_receiver.lua               ← owns :7000
  bridge_controller_player.lua    ← per-player manual control
tools/                   ← operator + dev helpers (see table above)
update-feed/             ← Cloudflare Worker + dashboard PWA
```

Sync flow: `npm run sync:shared` in any project copies `shared/*.js` into that project's `src/`. Always run before building.

Quick start (build and install)

Install (operator)

1. Download the latest `ConstantiaBSWall-Setup-<ver>-x64.exe` (NSIS) from this repo's **Releases** — or the `-Portable-` `.exe` for no-install use.
2. Run it. The installer detects any prior version (incl. the legacy machine-wide "ghost" install), offers back-up-and-clean vs install-over, and adds the Windows Firewall rule for the bridge (TCP 5000). May prompt for elevation only for the firewall step.
3. Launch from the Start Menu / desktop shortcut. The embedded bridge starts on `127.0.0.1:5000` automatically; Q-SYS reaches it over the LAN at `http://<laptopLanIp>:5000`.

Develop

```
# Wall
cd bs-provisioner
npm install
npm run sync:shared
npm start              # launches in dev mode

# Provisioner
cd bs-provisioner-classic
npm install
npm start
```

Build

Release artifacts come from **GitHub Actions** — push a tag and the workflow builds and attaches the assets:

Tag pattern	Workflow	Builds
`wall-v*`	`build-wall.yml`	Windows NSIS Setup + portable `.exe` (+ macOS `.dmg` when enabled)
`tools-autostart-v*`	`build-autostart.yml`	Q-SYS Designer auto-launch bundle (`.exe` + `.ahk` + `.ps1`)

Each build job runs `sync:shared` + `build:icon` as preflight, then `electron-builder`. A final `prune` job keeps only the latest release per tag-family.

> **GitHub Actions budget note:** the free-tier minutes were exhausted by the macOS matrix (macOS runners cost 10×). The on-site build path is now the canonical one:
>
> ```
> :: On-site Windows PC:
> cd <repo>
> tools\update-and-build.bat       :: git pull + npm install + sync:shared + electron-
>         builder NSIS + opens dist\
> ```
>
> Mac dev still works for code review + local testing (`npm start`), but the installer is built on the site PC.

Local dev build (smoke-test only):

```
cd bs-provisioner
npm ci
npm run build:nsis        # or build:portable / build:dmg
# Output: dist\ConstantiaBSWall-Setup-<ver>-x64.exe
```

> **Pre-req on a local Windows build machine:** Developer Mode = ON (Settings → System → For developers). Without it `electron-builder` fails to extract the `winCodeSign` cache (symlink permission error).

In-app updates and dashboard

In-app updates (private-repo-safe)

The Wall shows a non-blocking **"Update available — Download / Later"** banner on launch when a newer build exists. The source repo stays **private**; only a small `version.json` / `latest.yml` + the built installer are published publicly, so no code is ever exposed.

Flow: `build-wall.yml` (Windows job) builds the Setup `.exe`, attaches it to the private GitHub release, and — if Cloudflare is configured — publishes the feed + installer. The Wall fetches the feed (`VERSION_FEED_URL` in `bs-provisioner/src/main.js`), compares `latest` to its own version, and shows the banner. **Download** opens the public installer (NSIS upgrades in place); **Later** dismisses for the session.

Behaviour:

- **Dormant by default** — empty `VERSION_FEED_URL` (and no CF secrets) = no banner, no error.
- The feed URL is overridable at runtime via the laptop's `userData/update-config.json` (`{ "feedUrl": "https://…/version.json" }`) — no rebuild needed.
- `mandatory:true` in the feed would make the banner non-dismissable (current UX is notify + Later).

Hosted on the **studiouih** Cloudflare account: Cloudflare Pages caps files at 25 MiB, so the ~75 MB installer `.exe` goes to the **R2** bucket `constantia-updates` (served by the `constantia-dl` Worker), while `version.json` / `latest.yml` + docs go to **Pages** (`constantia-updates`). One-time setup (CF project, secrets, vars, `VERSION_FEED_URL`) is in `update-feed/README.md`.

Dashboard — `constantia-updates.pages.dev` (OTP-gated, installable PWA)

The Wall **pushes telemetry** to the `constantia-dl` Worker (KV-backed), and the Cloudflare Pages site renders it for the client (Iziko) + UWC. The repo stays private; only status data + the operator page leave. Installable to a phone home screen (PWA).

Login: email → 6-digit OTP (KV, 24 h Bearer session). Allowlist of operators; gated when the Worker var `REQUIRE_AUTH="true"`. To grant access: add the email to `ALLOW` in `update-feed/worker/index.js` + redeploy.

Tabs:

- **Logs** — boot & uptime history (times in SAST) + a Q-SYS audio detail table (every audio start/stop the master fired: buses fired/total, all-fired?, seconds after reboot, missing buses).
- **Live View** — each online player's latest snapshot (pushed with the heartbeat).
- **Ops** — **Show control** Play/Stop (drives the Q-SYS flip-flop via the command relay), a **Schedule** editor (enable, ON/OFF times in SAST, day toggles + Daily/Weekdays/Weekend presets), the current-status table, and **Remote access** (where the Windows login lives — see On-site PC).
- **Documentation** — embedded demo video; the operator `manual.pdf` / `manual.html`; the **Technical handover (full)** PDF (this README); and the **BrightAuthor presentation guide** PDF (`BUILD_A_PRESENTATION.md`).

How information reaches the dashboard (Core → dashboard). The dashboard never talks to the Core directly. The Wall (on the LAN) polls the Core over **QRC** (JSON-RPC :1710) and the bridge for projector/flip-flop/schedule state and Q-SYS audio events, then **pushes** that telemetry outbound to the `constantia-dl`

Worker (KV-backed); the Pages site reads the Worker's `*.json` and renders it. Marker/sync data originates on the Core's `sync_receiver` (the BrightSigns send `S0` / `L0` + `SYNC:<id>` straight to Core `:7000`); the Wall mirrors markers via the Core's display-only :7000 forward. So the chain is **BrightSigns → Core → (QRC/forward) → Wall → Worker(KV) → dashboard** — all outbound from the LAN, no inbound port, survives an internet drop (the Core keeps running the show).

How the players operate (no-reboot lifecycle). Boot → **Black/park** (silent, no markers). `09:55` schedule edge → Core flip-flop ON → `SendStart` (UDP) to all 6 → each player → **Show @ frame 0**, roSync locks (domain 6), Master emits `S0@0` → Core starts audio. `16:45` → flip-flop OFF → `SendStop` → all → **Black/park**, audio + projectors off. No daily reboot; `RebootPulse` is manual recovery only. Full authoring detail in `BUILD_A_PRESENTATION.md`.

Worker endpoints (outbound from the Wall, keyed `x-ingest-key`):

- `/ingest` — per-reboot boot history → `/log.json`; also fires a boot-status email to operators.
- `/heartbeat` — liveness + Q-SYS digest + snapshots → `/status.json`.
- `/qlog` — Q-SYS audio events → `/qlog.json`.
- `/command` — web Play/Stop mailbox (the Wall polls outbound, relays to the bridge).
- `/schedule` — schedule config store (the Wall relays it to the bridge → Lua; **no Worker cron** — the Core executes the schedule, so it survives an internet drop).

The schedule is a config relay, not a remote executor: the always-on Core polls and runs the last-known schedule even if connectivity drops.

Security notes

- **No device serials or passwords appear in this repo.** A BrightSign serial **is** its DWS password — real serials live only in the operator `.txt` configs pushed at runtime, and the Windows login lives in the gated dashboard (Ops → Remote access)
  - the operator's secure note. Do not paste any of these into git.
- **The dashboard** is OTP-gated (email allowlist → 6-digit code → 24 h session) when the Worker var `REQUIRE_AUTH="true"`. The public installer feed (`latest.yml` / `*.exe` / `*.blockmap` / `/download`) stays open so the auto-updater works. Telemetry writes use an embedded `INGEST_KEY` (drive-by write protection, not a real secret). Unsubscribe links are SHA-256-tokened.
- **DWS reboot is currently unauthenticated** because `legacyDWS=true` is the firmware default on this isolated LAN. If this install is ever placed on a mixed-trust network, set a DWS password via `PUT /api/v1/registry/networking/dwsp` (and update the Wall + Q-SYS Lua to send digest auth).
- **Both desktop apps are un-signed.** Windows SmartScreen will warn on first install. Sign with an EV cert for silent enterprise deployment.

Operational gotchas

System / sync architecture:

- **Only ONE component binds Core :7000** — `sync_receiver`. The master must have `SYNC_LOOP_ENABLE=false` or markers split between two listeners ("fires once then nothing").

- **F5/boot must be inert** — the master seeds-not-acts + has a 30 s reboot guard; each player has a 30 s guard. Q-SYS re-asserts wired pin **states** when a design reloads on Save-to-Core (F5); a latched flip-flop re-fires the Start net, which on the bridge **aliased to a reboot of the whole fleet**. The F5-reboot was a **per-player** `StartPlay` re-assertion from the flip-flop, not the master. Symptom that you're on an old Lua: projectors go off or the show fires on F5.
- **A player can't power itself off** (always-on). "Stop" = go to the `Black` state, not power-off. The video itself can't be stopped from Q-SYS — BA autostarts it.
- **Audio playing after a Stop/shutdown = the show-state gate (`showOn`).** The BrightSign videos keep looping after Stop (Stop doesn't reboot them) and every loop's L0 marker used to restart audio. Fixed by `showOn`: the gate closes after Stop/OFF → markers are logged but the audio drive is skipped. If audio still restarts, check the schedule tick log shows `audioGate=CLOSED`.
- **Wall :5000 relay re-forwards S0/L0 to :7000** — so the Core forwards markers to the Wall's **:7000** listener (display-only, never relays), NOT :5000. Feeding markers to Wall :5000 = a Core↔Wall infinite loop. (`sync_receiver.lua`'s `WALL_PORT` must stay 7000.)
- **Show didn't auto-start?** The baked schedule is 09:55–16:45 all days. If a load landed AT/after 09:55, the seed-not-act first tick swallowed it — the daily PC reboot must be ~15 min EARLIER (09:40) so the load finishes before the edge.
- **F5 from a Task Scheduler task does NOT land** (no foreground rights on the desktop — confirmed twice). Timed load = scheduled REBOOT → auto-login → shell:startup autostart F5. `schedule-qsys-load.ps1` is deprecated.
- **The Core clock drives `os.date`** — the Core's timezone must be Africa/Johannesburg (Core Manager + NTP), or the schedule fires at the wrong wall-clock time.

BrightSign / DWS:

- **Q-SYS Lua HTTP gotcha:** `HttpClient.Upload` is **POST/PUT only** (`Method="GET"` throws "*unsupported method : GET*"). Use `HttpClient.Download` for GETs.
- **Never hot-swap an SD card on a running BrightSign.** The kernel mount goes stale; the file API returns 500 with `EIO: i/o error, scandir '/storage/sd'`, the snapshot endpoint can't `mkdir remote_snapshots`, and the video pipeline can't read content → black HDMI. **Recovery:** `PUT /api/v1/control/reboot` forces a clean remount.
- **HDMI EDID handshake order matters:** display ON first, then BrightSign. If the EDID read fails the player falls back to 4096×2160×60p preferred mode and a 1920×1080 presentation can't render → black screen even though DWS reports fine.
- **Cards have a setup-then-presentation lifecycle.** First boot: `autorun.brs` (setup script) runs → swaps `pending-autorun.brs` (BA runtime) into place as `autorun.brs` → `RestartScript()` → presentation plays. Restoring from backup puts the card back in "fresh first-boot" state and you'll see the network-diagnostics screen again for ~60 s.
- **DWS file API quirk:** `PUT /api/v1/files/sd/<path>` creates a *directory* named `<path>`. To upload a *file*: `PUT /api/v1/files/sd/` (trailing slash, no name in path) with multipart form-data, field `file`, filename in the multipart `filename=` parameter.
- **Never replace the BA `autorun.brs` with a custom `Sub Main()`.** The supervisor watchdogs the BA runtime handshake → reset loop. Recovery: re-upload the stock BA `autorun.brs` (in git, backed up before any deploy) via DWS PUT `/api/v1/files/sd/`. Also: `roFileSystem` is Roku-only (panics on BrightSign), `roVideoPlayer`'s constructor takes no args, and `PlayFile()` takes an `roAssociativeArray`, not a string. Use BrightAuthor for content authoring, DWS for runtime control.
- **BrightAuthor classic has no CLI** (forum-confirmed) — only Sikuli-style GUI scripting works. Don't try to automate publishing.

Field troubleshooting runbook

On-site, fast-reference guide. Read the Network map and Quick checks first, then jump to the symptom that matches. Private / client deliverable — no device serials or passwords in this section; real serials live in the operator `.txt` configs pushed at runtime.

⚠️ ON-SITE RULES — READ BEFORE TOUCHING ANYTHING

The golden rule: on site you MONITOR and RECOVER. You do NOT PATCH. Every near-disaster in the field (nearly-bricked players, "dead" units, lost passwords, scrambled sync) came from editing the *running* system.

✅ **SAFE on site (read / recover, reversible):**

- Watch the Wall (status, screenshots, playback %).
- **Reboot** a player (Play/Reboot, or DWS `PUT /api/v1/control/reboot`).
- **Restore passwords** to the Wall bridge (PUT `/api/players/<id>`).
- **Clean net** / **Repair** buttons (clear gateway/divert + reboot).
- Read diagnostics, `curl` info, power-cycle.

🛑 **FORBIDDEN on site (do at the BENCH instead):**

- ❌ Wall → player **injection** / "Deploy BA bundle" to a live player.
- ❌ **Firmware flash** on the live rack.
- ❌ **Editing/replacing** a running player's video, autorun, or config.
- ❌ **Cancelling** a transfer mid-write (leaves a half-written SD → looks dead).

The safe way to change content/firmware = BENCH + SWAP: at the bench, prep a spare player + SD fully (firmware, video, config), verify it plays + syncs, **swap** the prepped unit (or just its SD) in on site and power on; if bad, swap the old one back. Reversible, zero live editing.

"Dead" player check (don't panic): an HTTP response — even **401** or **404** — means the player is ALIVE and answering. Truly dead = ECONNREFUSED / timeout / silence. `401` = "needs password" (use `--digest -u admin:<serial>`). `404` = "answered, wrong path". Neither is a brick. A cancelled injection / half-written SD just needs re-imaging at the bench — the hardware is fine.

Quick checks (run these first, every site visit)

From the Wall PC `Administrator: Command Prompt`:

```
:: Are all players reachable + what model/fw/uptime?
curl -s "http://192.168.1.11/api/v1/info"

:: Is a player's network config clean (no stray gateway)?
curl -s "http://192.168.1.11/api/v1/registry/networking/gw"
:: Expect {"value":""}. A non-empty gw on this LAN = trouble (see T-1).
```

On the **Wall dashboard**: each tile shows model / fw / uptime / playback % / live screenshot — these are the real health signals (not the diagnostics ping, see T-1). 📡 **UDP Listener** → 🔁 relay banner shows S0/L0 counts. 🩺 **Net Doctor** → PINGed/PONG, players, DWS fingerprint, sync markers.

T-0 · Player stuck / diverted / mis-networked — try Repair FIRST

One-click full auto-recovery (v0.11.15+): on the player's tile, click 🩹 **Repair**. It runs the common fix chain as one action with a single reboot:

1. clear the **divert-script flag** (so `autorun.brs` runs again next boot — inverse of the hammer),
2. clear any **stray gateway / DNS** (the diagnostics-timeout cause, T-1),
3. **reboot once** to apply both.

Safe on this isolated LAN. Use it as the first move for a player that's stuck, diverted after a hammer rescue, or showing a diagnostics timeout. If Repair doesn't recover it, fall through to T-1 / T-6. (Repair = 🧹 Clean net + clear-divert combined, with one reboot instead of two. Use 🧹 Clean net for the network cleanup alone.)

T-1 · Master tile shows "Timeout after Nms" on Network diagnostics

Cause: a stray **gateway** set on the device. BrightSign DWS `/api/v1/diagnostics` pings the gateway; if it's a dead IP (no such host on the isolated LAN), the ping hangs → the Wall times out. The Master (LS445) was imaged with `gw=192.168.1.2`; the followers (LS425) had none → only the Master hung.

Diagnose:

```
curl -s "http://192.168.1.11/api/v1/registry/networking/gw"
:: Master with the bug → {"value":"192.168.1.2"}   (a host that doesn't exist)
:: A clean player      → {"value":""}
```

Fix (clear the dead gateway + reboot):

```
curl -s -X PUT "http://192.168.1.11/api/v1/registry/networking/gw" -H "Content-Type:
        application/json" -d "{\"value\":\"\"}"
:: verify it took:
curl -s "http://192.168.1.11/api/v1/registry/networking/gw"        :: → {"value":""}
:: reboot to APPLY (registry network changes need a reboot):
curl -s -X PUT "http://192.168.1.11/api/v1/control/reboot"
:: after ~40-60s, re-verify it persisted:
curl -s "http://192.168.1.11/api/v1/registry/networking/gw"        :: → {"value":""}
```

If gated (401), add `--digest -u admin:<serial>`. Clear `d1` / `d2` (DNS) the same way if set.

One-click fix (v0.11.14+): on the player's tile, click 🧹 **Clean net** — reads gw/d1/d2, clears any stray value, and reboots to apply. Use this instead of the manual curls when on the Wall.

"It still shows the old result after I fixed the player" — it's the DEVICE, not the Wall. The Wall does NOT cache diagnostics (v0.11.21 adds `no-cache` headers + a `?_ts=` buster). BrightSign only **re-runs** its network self-test at boot / network change and serves the stored result on every GET. So after clearing a stray gateway/DNS **without rebooting**, the device keeps reporting the old failing test — **reboot to refresh it** (that's what 🧹 Clean net / 🩹 Repair do). The Master's endpoint is a separate case: it hangs entirely on the gateway-less LAN, so its "Timeout" is expected and benign regardless (the Wall shows it grey).

Suspected origin: the divert-script **hammer** (12-way parallel registry PUTs during a reboot-loop rescue) churns the networking registry; combined with the Master's different image, a stray gateway can end up written. If you've recently hammered a player, check its `gw` after.

⚠️ Rebooting the Master interrupts audio-sync briefly (it's the Sync Leader). Do it when the gallery is quiet.

T-2 · Audio out of sync with the Master's video

Design: Q-SYS holds the audio (Audio Players → bus → amp), the Master holds video; they drift ~80–160 ms/loop. The Master's BA fires `S0` (frame-0-ish) and `L0` (each loop end); the Core re-seeks the Q-SYS audio to match.

Check the chain (in order):

1. **Player sends markers?** Wall → 📡 UDP Listener. Look for **2-byte** packets from `192.168.1.11` with payload `S0` / `L0`. (Old bug: 17-byte packets with payload `192.168.1.10:7000` = the IP was typed into the message field. Fix in BA: the Send-UDP message field must be the literal `S0` / `L0`; delete any "Send UDP bytes" command.)
2. **Markers reach the Core?** Under the no-reboot design they go to Core `192.168.1.10:7000` DIRECT. (On the historical relay path they went to the Wall :5000, which relayed to the Core — watch the 🔁 relay banner.)
3. **Core re-syncs audio?** Core Event Log / `SyncLastMarker` → `recv L0 … → re-sync audio`; `AudioPlayState` → `N/16 playing`.

Test without waiting the 13:36 loop:

```
node tools/sync-fire.js --profile constantia --loops 3 --gap 6
:: fires S0 then 3× L0 at the Core.
```

BA event setup (Master only): `S0` = Video Timecode @ 0 ms (or historical Timeout @ 0.10 s), Send UDP msg `S0`; `L0` = Media-End event, Remain/Continuous, Send UDP msg `L0`. The Lua's offset must match the Timeout value if you use the Timeout form. Followers do NOT send markers.

T-3 · "Are the audio buses actually playing?" (not just "did we fire")

Old bug: the Lua reported "16/16 OK" meaning *commands sent*, not *playing*. Fixed in v2.20.12: it reads back state and reports **confirmed playing**.

Check:

- Core log: `audio start: sent 16/16, CONFIRMED playing 14/16 | NOT playing: …`
- Optional `AudioPlayState` Text control (Script Access = Script) → live `N/16 playing @ HH:MM:SS` (polls every 10 s).
- Wall dashboard audio line → `… · live: 14/16 playing`.

If it says `0/16` or many `state-unreadable`, capture the exact control name (right side of a player in Q-SYS) and report it. Note: `play.state=true` **resumes** from the paused playhead — it does NOT restart from 0. Re-sync seeks to 0 first (or falls back to stop→start, which rewinds).

T-4 · Projector / component state not showing on dashboard

Old bug: the bridge collapsed any non-"event" kind into "audio", mangling `kind:state` (projectors). Fixed in wall-v0.11.12. If states still don't show:

- Confirm the Q-SYS component has **Script Access = Script** (or All), else `Component.New` returns nil → boot log says `watcher MISSING`.
- `WATCHERS` ships pre-filled with the 5 Epson projectors (v2.20.11): `EpsonProjector`, `EpsonProjector_1`..`_4` on `power.state`.
- Dashboard → "Q-SYS state changes" table populates on each change.

T-5 · Player won't follow the leader (video) — only relevant for a video WALL

Separate rooms → ignore this (see The two sync systems). For a tiled wall: all 6 on Player Sync **Enable**, same **Domain** (6), exactly **1 Leader**, rest **Followers**. Frame-lock needs **frame-exact equal duration + same fps** (file *size* and *name* are irrelevant). net-doctor "roSyncManager Leaders: none announced" is a false negative on fw 9.x — trust your eyes. Must **republish + redeploy all 6** after changing sync settings (editor config ≠ deployed).

T-6 · Master reboot loop / black screen

Never replace the BA `autorun.brs` with a custom `Sub Main()` — the supervisor watchdogs the BA runtime handshake → reset loop. Recovery: re-upload the stock BA `autorun.brs` (in git, backed up before any deploy) via DWS PUT `/api/v1/files/sd/`, or use the Wall's restore-stock / SD-wipe / divert-script hammer buttons.

"Congratulations, your BrightSign player is set up!" splash + never plays: BA was published as "Setup files only". Fix: BA:Connected → Destination Type → **Standalone / Local Storage** → republish (should give ~1 MB autorun.brs, no setupCommon / pending-autorun).

T-7 · One player drifts while the rest stay locked (roSync)

The single biggest sync gotcha (field 2026-05-31). All players on the same firmware, Domain 6, 1 Leader — yet the Master drifted while the 5 followers stayed locked. Cause was NOT network, NOT resolution, NOT firmware.

Cause: unequal LOOP LENGTH (frame count). roSync frame-locks; if one clip has more frames, it falls behind by the difference EVERY loop and drifts forever. Field values: 5 followers' clips = **20402 frames** @ 25 fps (816.1 s) → locked; the Master's `Office1_RotatedVideo` = **20494 frames** (819.8 s, ~3.6 s longer) → drifted ~3.6 s/loop.

Diagnose — probe the frame count of every clip (count, don't trust duration):

```
ffprobe -v error -select_streams v:0 -count_frames \
  -show_entries stream=width,height,r_frame_rate,nb_read_frames \
  -of default=nw=1 "clip.mp4"
# All clips MUST report the SAME nb_read_frames and the SAME r_frame_rate.
```

Ruled out on site: resolution (followers ran a mix of 1762×848 and 1920×1080 and locked fine), network (gateway/subnet, cleared, fine), and firmware (once all 6 matched 9.1.93.2, still drifted until the frame count was fixed).

Fix — trim the long clip to the common frame count (see the `ffmpeg` command in roSync setup). Or batch all clips with `tools/encode-videos.sh`.

Firmware parity caveat: OTA firmware updates can silently fail validation and leave the player on the OLD version (the file ends up renamed `cobra-update.bsfw_invalid` on the SD). The DWS INFO `FWVersion` is ground truth — check it, don't assume the update took. If it won't update OTA, flash via SD recovery.

Deploying a changed video via DWS — the manifest trap: see the `pool/` + sha1 + `local-sync.json` note under Part C — deploy. Uploading just a new `.mp4` does nothing — re-publish in BA (Standalone) so the manifest regenerates, then push the whole folder with 🔧 **Deploy BA bundle**.

T-8 · Wall shows players "unreachable" but they're actually fine (lost passwords)

Symptom (field 2026-05-31): players show red/unreachable on the Wall; the Import-configs button auto-cancels; a bare `curl` returns **401**. The players are FINE and playing — the Wall just lost the stored DWS passwords, so it can't authenticate to the gated players (.30–.34).

Confirm the players are alive (with auth):

```
curl -s --digest -u admin:<serial> "http://192.168.1.31/api/v1/info"
:: Returns full info → alive. (401 on a bare curl = alive + gated, NOT dead.)
```

Fix — re-store the passwords directly in the bridge (no Import needed). The players already exist in the bridge (POST returns 409), so UPDATE each by id with PUT. Get the real ids first:

```
curl -s "http://127.0.0.1:5000/api/players"        :: lists id + ip for each
```

Then one PUT per player (DWS password = the player's serial unless changed):

```
curl -s -X PUT "http://127.0.0.1:5000/api/players/<id>" -H "Content-Type: application/json" -d
       "{\"password\":\"<serial>\"}"
```

Refresh the Wall → tiles go green. (Serials live in the operator `.txt` configs, not in this repo. The Import-configs auto-cancel is a Wall bug to be fixed at the bench, not on site.)

Key commands cheat-sheet

```
:: --- player info / health ---
curl -s "http://192.168.1.<N>/api/v1/info"

:: --- network registry (gateway/dns/dhcp/ip) ---
curl -s "http://192.168.1.<N>/api/v1/registry/networking/gw"
curl -s "http://192.168.1.<N>/api/v1/registry/networking/dhcp"
curl -s "http://192.168.1.<N>/api/v1/registry/networking/sip"

:: --- clear a stray gateway (see T-1) ---
curl -s -X PUT "http://192.168.1.<N>/api/v1/registry/networking/gw" -H "Content-Type:
        application/json" -d "{\"value\":\"\"}"

:: --- reboot a player ---
curl -s -X PUT "http://192.168.1.<N>/api/v1/control/reboot"

:: gated player? add:  --digest -u admin:<serial>
```

```
:: --- audio-sync test (from repo root, no wait for the real loop) ---
node tools/sync-fire.js --profile constantia --loops 3 --gap 6
node tools/sync-fire.js --token L0          :: one L0 and exit
node tools/sync-fire.js --token S0          :: one S0 and exit

:: --- LAN / sync diagnostic ---
node tools/net-doctor.js --profile constantia --sync-secs 120 --reboot-first
```

Versions and changelog

- **Wall:** see `bs-provisioner/package.json`. Releases tagged `wall-v<x>`.
- **Q-SYS bridge controllers:** tagged `bridge-controller-v<x>`.
- **Autostart tools:** tagged `tools-autostart-v<x>`.
- **Provisioner:** see `bs-provisioner-classic/package.json`. Releases tagged `provisioner-v<x>`.

The version table below and `CHANGELOG.md` are auto-generated from git tags by `.github/workflows/update-docs.yml` (runs `tools/gen-docs.js` on every push to `main`). See `CHANGELOG.md` for the full per-tag history.

Component	Latest	Date	Tag
Constantia BrightSign Wall	`0.11.24`	2026-06-02	`wall-v0.11.24`
Q-SYS bridge controllers (Lua)	`2.20.7`	2026-05-29	`bridge-controller-v2.20.7`
Q-SYS Designer autostart bundle	`1.5.0`	2026-05-22	`tools-autostart-v1.5.0`
Constantia BrightSign Provisioner	`0.2.0`	2026-04-30	`provisioner-v0.2.0`

Updated 2026-06-02 from 162 tags.

Local-only dependencies (not in repo)

- **Master SD card backup** (~782 MB, includes the client's video) — byte-for-byte backup of the original master SD card. Used by `bs-provisioner-classic` as the clone source. Excluded from git via `.gitignore`. Recreate from the live master card if lost.
- **Q-SYS Designer installer** (`Q-SYS_Designer_Installer_*.zip`) — from QSC's qsys.com (account required), downloaded once per machine.
- **`node_modules/`** — `npm install` in each project folder.
- **Video masters** — `…/GrootConstantia/VIDEOs/` (the source clips; the published `pool/` media is rebuilt from these and is not committed).

Related documents

Sub-component docs (kept separate; not merged here):

- `BUILD_A_PRESENTATION.md` — step-by-step BrightAuthor:Connected authoring guide + lessons-learned for the 6 presentations (states, roSync, UDP, markers, sync heartbeat, publish, verify, traps).
- `CHANGELOG.md` — full per-tag history (auto-generated).
- `bs-provisioner/README.md` — the Wall app in detail.
- `bs-provisioner/BUILD-WINDOWS.md` — Windows build notes.
- `bs-bridge/README.md` — standalone CLI bridge.
- `qsys-bs-bridge/README.md` — Q-SYS Lua controllers.
- `update-feed/README.md` — Cloudflare Worker + dashboard setup (CF project, secrets, vars, `VERSION_FEED_URL`).

Built and supported by the UWC Immersive Zone.