VTX Media 1: audio and video

Nov 15, 2026 · VTX 1 · extension · draft d02 · MIT licence · vtx.foundation/spec/v1/media-1/

Media 1 adds audio and video to VTX: Opus audio and AV1, H.264, HEVC or VP9 video on dynamic channels in both directions, live media on QUIC datagrams, a `media` node and a `pixel` surface in the scene, and microphone, camera and screen capture behind trusted-zone permissions.

§1 Status and scope

Media 1 is draft d02 of an extension of VTX 1, in conformance profile 3, built on the core at draft d03; d02 adds keyboard lock. It fills the core's extension points and adds two dynamic channel types; no rule of the core changes. It adds:

- `audio` and `video` channels in both directions: *play* (server to client) and *capture* (client to server);
- live media on QUIC and WebTransport datagrams (RFC 9221), and buffered media on the channel's own stream under v1 credits;
- a `media` node with the client's own playback controls, a `pixel` surface that shows a video stream and takes input, and a `sound` op for short cached clips;
- clip resources (Ogg Opus, WebM, MP4) on the resource channel;
- the `microphone`, `camera` and `screen` permissions, with a capture indicator in the trusted zone, and `keyboard.lock` for remote desktops and terminals;
- application lines, so an agent can speak and listen without touching the wire.

Opus is required for audio. A client that decodes video `MUST` decode H.264 or AV1, and a server that sends video `MUST` be able to encode both; HEVC (H.265), VP9 and PCM are optional.

Builds on. The `pixel` surface is a Surfaces 1 surface and reuses its `sx`, `sy` pointer fields, so a client that offers `surface.pixel` also offers Surfaces 1. Everything else needs v1 only.

Not in Media 1. Window surfaces (RemoteApp parity) are Windows 1, with several monitors in Displays 1. Calls between attached clients come with multi-viewer, where WebRTC stays optional. More than one audio track per stream, forward error correction and DRM are left out.

§2 Capabilities and limits

A client offers what it can play and capture in HELLO and CAPS; the server's answer lists the keys it will use, as in v1. The keys sit under the `audio`, `video` and `surface.` prefixes that v1 reserved.

Key	Value	Meaning
`audio`	`codecs` (`MUST` include `opus`), `channels` (1–8, default 2), `captions`	Plays audio streams. `captions: true` means the user wants CUE frames shown
`audio.capture`	`codecs`, `channels` (1–2), `processing` (`echo`, `noise`, `gain`)	Has a microphone it can capture, and the clean-up it can apply

Key	Value	Meaning
`video`	`decode`: a list of `codec`, `max` [w, h], `fps`, `bits` (8, 10), `chroma` (420, 444), `hw`	Decodes video streams; the list `MUST` include `h264` or `av1`
`video.capture`	`sources` (`camera`, `screen`), `encode`: the same list form	Can capture a camera or its screen
`audio.clips`, `video.clips`	Media types	Plays clip resources of these types
`surface.pixel`	`true`	Shows a video stream in a `pixel` surface (needs `video` and Surfaces 1)
`input.keylock`	`true`	Can capture system shortcuts for a node with `keys: "all"` (Section 7)

Three limits join HELLO and CAPS. They bound what a hostile or careless server can make the client decode.

Limit	Default	Bounds
`media_kbps`	8,000	All media the client receives, in kbit/s
`media_channels`	8	Audio and video channels open at once
`video_px`	8,294,400 (3840 × 2160)	Coded pixels in one video frame

When a server breaks a media limit, the client closes that channel with CLOSE `limit_exceeded` (0x0A); the rest of the session goes on.

§3 Media channels

Media travels on two dynamic channel types, `audio` and `video`. Only the server opens them, with OPEN on channel 0 and an even ID from 16 (v1 Section 4); the client answers OPEN_OK or OPEN_REFUSED. Each channel carries one track in one direction:

- **play** channels carry media from the server to the client;
- **capture** channels carry the client's microphone, camera or screen to the server, once the user has granted the permission (Section 8).

Streams. OPEN names a stream. A stream has at most one audio and one video channel, in one direction, on one clock, so the client keeps them in sync. The scene and the application refer to streams by name, never by channel ID; after a resume the server opens new channels under the same names and every binding still holds.

OPEN param	Play	Capture	Meaning
`stream`	Required	Required	Stream name, 1–64 of `A–Z a–z 0–9 _ . : -`
`dir`	`play`	`capture`	Direction
`codec`	Required	—	Codec family, from the client's `audio` or `video` capability
`codecs`	—	Required	Families the server accepts, preferred first; the client's CONFIG names its pick

OPEN param	Play	Capture	Meaning
`mode`	`live` or `buffered`	— (always live)	See the next table
`source`	—	`microphone`, `camera` or `screen`	What to capture
`delay`	Optional	—	Playout delay the sender intends, ms
`kbps`	Optional	Optional	Expected bitrate (play) or target (capture)
`ms`, `title`	Optional	—	Duration of buffered media, and a title
`rate`, `channels`, `processing`	—	Optional	The microphone audio wanted
`max`, `fps`	—	Optional	Largest video size and frame rate wanted

	Live	Buffered
Typical use	Remote desktops, calls, agent voice, cameras	Recordings, long speech, podcasts
MEDIA travels on	Datagrams; the channel's stream only on a fallback transport	The channel's stream
Loss	Possible; NACK and key-frame requests repair it	None
Flow	Rate control (Section 5)	v1 credits, or QUIC stream flow control: a paused client stops granting window
Seek and end	—	MARK
Playout delay	`delay`, often 0–100 ms	The client's choice, often seconds

Datagrams.

- A QUIC or WebTransport datagram (RFC 9221) carries exactly one v1 frame: Channel ID, Frame Type, Length and Payload, with Length equal to the bytes that remain.
- Only MEDIA and FEEDBACK of live channels travel as datagrams, and on a connection that has datagrams, live MEDIA always does. CONFIG, MARK, CUE, STATE and VIEW always use the channel's stream.
- A peer that accepts Media 1 on QUIC sends the `max_datagram_frame_size` transport parameter. A sender keeps each datagram within the largest payload its QUIC stack reports for the path, a little under 1,200 bytes on a minimum-size path, and splits larger frames into parts (Section 5).
- Datagrams take no credit and QUIC never retransmits them, but they are congestion-controlled.
- A receiver silently drops a datagram for a channel that is not open, or of another frame type, because datagrams can overtake OPEN_OK and CLOSE. It holds a MEDIA datagram for up to the playout delay while the CONFIG it needs is still on its way.

Stream transports are the fallback. Live video over TCP is allowed only when there is no datagram route. A client that offers `video` tries QUIC first, or WebTransport in a browser, and uses TLS over TCP or WebSocket only when it cannot reach the server over UDP; a QUIC peer that sends no `max_datagram_frame_size` counts as no route too. SSH is chosen for itself and lands here as well. On the fallback, live media rides the channel's stream. Live audio plays with a larger jitter buffer. A live video sender keeps its queue short: it lowers the bitrate or skips frames before encoding when the window or socket stays full, and never drops an encoded

frame that later frames depend on unless a key frame follows. A client on the fallback SHOULD try QUIC again at its next reconnect, and can move the session there by resuming it. Serial links carry no media, and clients on them do not offer the capabilities.

Priority. v1 sends dynamic channels last. Media 1 senders SHOULD use this order instead: control; input and zone; live audio; scene; live video; resources; clipboard, buffered media and other dynamic channels.

Detach and resume. DETACH or a lost connection closes every media channel. Capture stops at once and never continues while no client is attached. After a resume the server reopens what it still needs: live streams continue from now, and buffered streams from wherever the application chooses, since it hears of the resume as a session event.

§4 Frames

Audio and video channels share seven frame types. As on the well-known channels, the types are scoped to the channel type; CREDIT and CLOSE work as in v1. The *sender* is the server on a play channel and the client on a capture channel.

Type	Frame	Sent by	Travels on	Fields	Purpose
0x10	CONFIG	Sender	Stream	`cfg`, `codec`, `rate`, `channels`, `w`, `h`, `fps`, `color`, `range`, `desc`	Describes the bitstream, before the first MEDIA and whenever it changes
0x11	MEDIA	Sender	Stream or datagram	First part: `n`, `pts`, `dur`, `key`, `cfg`, `parts`, `data`. Later parts: `n`, `part`, `data`	One encoded frame, or one part of it
0x12	FEEDBACK	Receiver	Stream or datagram	`rx_kbps`, `max_kbps`, `loss`, `jitter`, `delay`, `late`, `key`, `nack`	Reception report, bitrate cap, key-frame and resend requests
0x13	MARK	Sender, buffered only	Stream	`pos` or `end`	A new segment after a seek, or the end of the media
0x14	CUE	Sender	Stream	`pts`, `end`, `text`, `lang`, `speaker`	A caption on the stream's clock
0x15	STATE	Client	Stream	`state`: `active`, `muted` or `blocked`	What the user or the platform did locally
0x16	VIEW	Receiver, video only	Stream	`px`, `visible`	The size the video is shown at (play) or wanted at (capture), and whether it is on screen

- A frame from the wrong side, or on the wrong channel type, is a channel error: CLOSE with `protocol_error` on that channel.
- `cfg` numbers a channel's CONFIGs from 0. MEDIA without `cfg` uses the latest one. The first video frame after a CONFIG is a key frame.
- `color` defaults to `bt709` and `range` to `limited`; screen content SHOULD be sent as `srgb`, `full`. `desc` is only for Opus with more than two channels: the OpusHead packet (RFC 7845, Section 5.1).
- STATE `muted`: the user muted this stream (play) or this source (capture). While a capture is muted the client sends no MEDIA on it. `blocked`: the platform holds audible playback until a user activation. `active`: either has ended.

- VIEW `visible: false` means the video is off screen, behind a hidden tab or a scrolled-away node. The sender SHOULD pause video until it is visible again, and keep the audio going.

§5 Timing and adaptation

Every MEDIA carries `pts` in microseconds. The audio and video of one stream share a clock, and capture uses the client's input clock, so speech lines up with pointing.

Clocks.

- `pts` and `dur` are microseconds. `pos` (MARK and the `media.seek` intent) and `ms` are milliseconds, because people see them.
- On a play stream the server picks the clock's origin; the stream's audio and video share it.
- On a capture stream, `pts` is the client's monotonic input clock, the one input events use for `t`. An agent can match "this one" in speech with the pointer event at the same moment.
- The receiver plays audio as the master clock and shows each video frame when the audio reaches its `pts`. For live streams it keeps the playout delay near `delay`, adapting to the jitter it measures.

Frames and parts.

- `n` numbers a channel's frames from 0 and rises by one per frame. Video frames go in decode order, so video `pts` may step back for reordered frames; audio `pts` never does.
- A frame bigger than a datagram, or than the receiver's `max_frame` on a stream, is split. The first part carries the frame's fields and `parts`; the others carry `n`, `part` (1 to `parts` − 1) and `data`. A sender sends the parts in order, back to back on their channel.
- Where to cut is up to the sender. `vectors.json` shows one split: the 9,371-byte AV1 key frame of the example stream becomes eight datagrams of at most 1,200 bytes, the limit the vector sets.

Loss and repair (live channels).

- A gap in `n`, or a frame still missing parts when its playout time comes, is a loss. The receiver drops the frame and conceals it: Opus has packet loss concealment, and video waits for the next frame it can decode.
- FEEDBACK `nack` lists the `[n, part]` pairs still worth having. The sender MAY resend them unchanged if they can still arrive in time.
- FEEDBACK `key: true` asks for a key frame. The sender answers at most once per round trip. On datagrams it SHOULD also use periodic intra refresh, or a key frame every few seconds, so a receiver recovers even when its requests are lost.
- A receiver sends FEEDBACK at least every 200 ms while live media flows.

Bitrate.

- All media to a client stays within `media_kbps`, and each channel within the latest FEEDBACK `max_kbps`.
- On QUIC, a sender SHOULD set its target from the connection's congestion state and the `delay` and `loss` FEEDBACK reports, with a delay-based controller such as SCReAM (RFC 8298) or NADA (RFC 8698).

- When bandwidth runs short, video gives way first: audio keeps its rate while video lowers resolution, frame rate or bitrate. VIEW tells the sender the size the video is really shown at, so a thumbnail never gets a 4K stream.

Buffered media. Buffered channels need no rate control: v1 credits, or QUIC stream flow control, pace them. The client grants window as it plays, so pausing stops the flow by itself. After a seek the server sends MARK with the new `pos`; the client drops what it buffered and plays from the frames that follow. MARK `end` ends the media, and the client then sends the `media.ended` intent for the node that played it.

§6 Codecs

Every client that plays audio decodes Opus; video uses H.264 or AV1 everywhere, and HEVC (H.265) or VP9 where both sides have them. Every bitstream carries its parameters in band, so a receiver can join at any key frame.

Family	Required	CONFIG codec	One MEDIA frame holds	Notes
`opus`	With `audio`	`opus`	One Opus packet (RFC 6716), 2.5–60 ms	48 kHz clock; 20 ms packets are usual, 10 ms for calls
`pcm`	No	`pcm`	Interleaved 16-bit little-endian samples	8, 16, 24 or 48 kHz; for local links and simple devices
`av1`	`av1` or `h264` with `video`	`av01.P.LLT.DD`, e.g. `av01.0.08M.08`	One temporal unit, low-overhead OBU format, temporal delimiter first	A sequence header OBU in every key frame. Its screen content tools suit remote desktops
`h264`	`av1` or `h264` with `video`	`avc1.PPCCLL` or `avc3.…`, e.g. `avc1.640028`	One access unit, Annex B byte stream	SPS and PPS before every IDR. Decoded in hardware almost everywhere
`hevc` (H.265)	No	`hvc1.…` or `hev1.…`, e.g. `hev1.1.6.L120.B0`	One access unit, Annex B byte stream	VPS, SPS and PPS before every IRAP picture
`vp9`	No	`vp09.PP.LL.DD`, e.g. `vp09.00.40.08`	One frame or superframe	

- Codec strings follow RFC 6381 as the W3C WebCodecs codec registry lists them, so a browser client hands them straight to `AudioDecoder` and `VideoDecoder`.
- The `avc1`/`avc3` and `hvc1`/`hev1` prefixes do not change the format. The bitstream is always Annex B with parameter sets in band, so CONFIG needs no description record.
- For remote desktops, a server `SHOULD` send 4:4:4 chroma (AV1 High profile, H.264 High 4:4:4, HEVC range extensions) when the client's `decode` entry lists `444`, so small text stays sharp.
- The example streams read as `av01.0.00M.08` and `avc1.64000D`; the validator derives both strings from the bitstreams themselves.

§7 Scene

Three additions bring media into the scene: a `media` node with the client's own controls, a `pixel` surface for remote desktops and other live pictures, and a `sound` op for short clips.

Media node.

```
{"op":"open","id":"brief","type":"media","parent":"room","stream":"briefing","alt":"Morning
briefing, 4 minutes, audio"}
```

Prop	Meaning
`stream` or `res`	Exactly one: a stream name, or a clip resource the client plays from its cache
`alt`	Required text alternative; also what a client without the media shows
`controls`	Show the client's play, pause, seek and volume controls (default `true`)
`autoplay`, `muted`	Start playing; start muted
`loop`	Clips only
`poster`	An image resource, shown before playback and for audio-only media
`size`, `fit`	As on surfaces

- The controls belong to the client. Volume and mute stay local; STATE `muted` tells the server.
- Play, pause and seek reach the application as intents on the node: `media.play`, `media.pause` and `media.seek` (with `values.pos` in ms), plus `media.ended` when the media ends. The client pauses its own playout at once rather than waiting for the application. For a clip it also plays and seeks locally.
- A media node bound to a capture stream shows the client's own preview, drawn locally: a camera self-view.
- One node at a time shows a stream; a second binding is scene error `invalid_op`.

Pixel surface. A Surfaces 1 surface of kind `pixel` shows a video stream and takes input, as remote desktops and legacy GUI applications need.

```
{"op":"open","id":"desk","type":"surface","kind":"pixel","parent":"room","stream":"desk","alt":"Remote
desktop of build-01","size":
[1280,720],"fit":"contain","mouse":"move","cursor":"default"}
```

- POINTER and GESTURE over it carry `sx`, `sy` in the stream's pixels, as the latest CONFIG's `w` and `h` define them. Keys reach it while it has focus, as with any node, and `pointer: "lock"` gives the relative motion games need.
- `cursor` is the remote pointer, drawn by the client so it moves without delay: a pointer shape such as `text`, or `{"res": <image>, "hot": [x, y]}`. The server patches it whenever the remote cursor changes.
- When VIEW reports a new size, a remote desktop server `MAY` resize the desktop to match and send a new CONFIG and key frame, as RDP's display control does.
- `draw` and `edit` stay vector-only.

Keyboard lock. Remote desktops and terminals need keys the client's platform would otherwise keep: Alt+Tab, the Windows or Command key, Ctrl+W, Alt+F4. A grid or pixel surface with `keys: "all"` asks for them; `raw` is the default and `none` takes no keys.

- On the user's first click in the node, the client asks for the `keyboard.lock` permission in the trusted zone (capability `input.keylock`).

- While the node has focus and the grant holds, the client captures what its platform allows and sends it as KEY events. A browser can do this only in fullscreen, through the Keyboard Lock API.
- The reserved key combination is never captured and always releases the lock. Combinations the operating system keeps, such as Ctrl+Alt+Del, stay with it; a client `MAY` offer to send one from its own controls.
- Without the grant, `all` behaves as `raw`.

Sound. `{"op":"sound","res":"sha256:…"}` plays a short clip once. With `id` a sound can be stopped (`{"op":"sound","id":"ring"}`) or replaced; `loop` needs `id`, and `volume` runs from 0 to 1. Sounds are not scene state: a resumed client never plays sounds from states it missed, and a muted or blocked client drops them rather than queueing them.

Clip resources. Clips are content-addressed resources on the resource channel: `audio/ogg` (Ogg Opus, RFC 7845), `audio/webm`, `video/webm` and `video/mp4`, with tracks in Media 1 codecs. OFFER's `meta` gives `ms`, plus `w` and `h` for video, so a client can refuse before downloading. The client plays clips itself, with full seeking and no round trip. The example clip, `chime.opus`, is 2,672 bytes and 600 ms long.

§8 Capture and privacy

The server can only ask for capture. The user decides in the trusted zone, sees it there for as long as it runs, and can mute or stop it there.

- **Permissions.** `microphone`, `camera` and `screen` join v1's permission registry. The server asks with PERMISSION on the zone channel and the user answers in the trusted zone, with v1's scopes `once`, `session` and `always`.
- **Grants.** A server opens a capture channel only while it holds a grant for the channel's source. A `once` grant covers one channel; `session` and `always` grants cover more. Without a grant the client answers OPEN_REFUSED `refused`.
- **Indicator.** While a capture channel is open, the trusted zone shows each live source with mute and stop controls. The server cannot draw over it, theme it or hide it, and no capture runs without it.
- **Mute and stop.** Mute sends STATE `muted` and stops MEDIA on that channel. Stop closes the channel with CLOSE `user_stopped`, close code 0x0E, which Media 1 adds.
- **Screen.** The user picks the screen or window in a picker the client owns. The server never names what to share, and learns only the size of the result from CONFIG.
- **Processing.** The server may ask for echo cancellation, noise suppression and gain control; the client applies what it can before encoding. Echo cancellation matters for voice agents, whose own speech plays from the same device.
- **Little metadata.** The server learns the source kind, never device names. Detach stops every capture (Section 3).

  Diagram: capture flow · permission, indicator, mute and stop. Drawn in the PDF version, `04-media-1.pdf`.

The server only asks and opens. The answers, the mute and the stop come from the trusted zone, which shows the indicator for as long as the capture channel is open.

§9 Application interface

An application speaks and listens in JSON lines; the server runs the channels, transcodes audio the client cannot take, and keeps the timing. `vtx-app-media-1.schema.json` and `vtx-app-events-media-1.schema.json` define every line.

From the application.

Line	Fields	What the server does
`stream` with `dir: play`	`name`, `media`, `codec`, `mode`; `rate`, `channels`, `w`, `h`, `fps`, `kbps`, `ms`, `title`	Opens a play channel to the client, and again after every resume
`stream` with `dir: capture`	`name`, `media`, `source`, `codec`; `rate`, `channels`, `processing`, `max`, `fps`, `kbps`	Opens a capture channel, once the source's permission is granted
`media`	`stream`, `track`, `pts`, `dur`, `key`, `base64`	Sends one encoded frame, or a chunk of PCM, on a play stream
`cue`	`stream`, `pts`, `end`, `text`, `lang`, `speaker`	Sends a caption as CUE
`mark`	`stream`, `pos` or `end`	Sends MARK on a buffered stream
`stream_close`	`stream`	Closes the stream's channels; ends a capture
`resource`	v1 fields, plus the clip formats	Offers a clip
`permission`	v1 fields, plus `microphone`, `camera`, `screen`, `keyboard.lock`	Asks the user in the trusted zone

To the application.

Line	Fields	Meaning
`media`	`stream`, `track`, `pts`, `dur`, `key`, `base64`	Captured media; `pts` is on the client's input clock
`stream_state`	`stream`, `state`, `reason`, `codec`	`open`, `active`, `muted`, `blocked`, `closed` or `refused`, with a reason: `permission`, `unsupported`, `user`, `limit`, `detached` or `ended`
`stream_feedback`	`stream`, `kbps`, `key`, `px`, `visible`	For an application that encodes video itself: target bitrate, key frame wanted, the size shown, whether it is on screen
`intent`	v1 fields	`media.play`, `media.pause`, `media.seek` and `media.ended` from media nodes

- `codec: pcm` lets an application work in plain samples both ways. The server encodes PCM to Opus for a client without PCM, and decodes captured Opus to PCM at the `rate` asked for. Video always arrives encoded; the server does not transcode it.
- `track` names `audio` or `video`, and is needed only when a stream has both.
- A video stream's first `media` line is a key frame, and every key frame carries its parameter sets: SPS and PPS, VPS too for HEVC, the sequence header for AV1.
- Audio `pts` never goes back. `mark` is for buffered streams only.
- The A2UI gateway maps A2UI's `Video` and `AudioPlayer` to media nodes. It fetches the URL itself, then offers a short file as a clip or serves a long one as a buffered stream, so the client still fetches nothing.

A voice agent, abbreviated (→ the application writes, ← the server writes):

```
→ {"kind":"permission","id":"mic-1","perm":"microphone","detail":"Talk to the
concierge"}
← {"kind":"zone_result","id":"mic-1","decision":"allow","scope":"session"}
→
{"kind":"stream","name":"mic","media":"audio","dir":"capture","source":"microphone","codec":"pcm","rat
["echo","noise","gain"]}
← {"kind":"stream_state","stream":"mic","state":"active"}
← {"kind":"media","stream":"mic","pts":81234567,"dur":5000,"base64":"AAAm…"}
→
{"kind":"stream","name":"voice","media":"audio","dir":"play","codec":"pcm","rate":24000,"channels":1,"
→ {"kind":"cue","stream":"voice","pts":0,"end":1800000,"text":"Your table for two is
booked for 19:30."}
→ {"kind":"media","stream":"voice","pts":0,"dur":5000,"base64":"AAAh…"}
```

§10 Fallbacks and security

A client without media still shows every media node as its `alt` text, and no capability lets a server capture, decode or spend beyond what the client allowed.

The client lacks	What happens
`audio`	No audio channel opens. Media nodes show `alt` and `poster`; the application hears `stream_state` `refused`, reason `unsupported`
`video` but has `audio`	Only a stream's audio channel opens; its media node shows the poster with audio controls
`surface.pixel`	The pixel surface shows its `alt`
`audio.clips` or `video.clips`	Clip media nodes show `alt`; `sound` plays nothing
`audio.capture` or `video.capture`	Capture streams are `refused`, reason `unsupported`
A datagram route (QUIC unreachable, or SSH)	Live media rides the stream as a fallback (Section 3)
All of it (serial links, grid-only clients)	Nothing opens; `alt` text stands in

Security.

- Decoders are the largest attack surface Media 1 adds. Clients `SHOULD` decode with platform or hardware decoders, or inside a sandbox, and check each CONFIG against `video_px`, the codec level and their own `decode` entry before feeding a decoder.
- The media limits bound what a server can make a client spend (Section 2). Clip size is bounded by `cache_bytes`, and OFFER's `meta` lets a client refuse a clip before downloading it.
- A client `MAY` hold audible playback until a user activation, as browsers do (STATE `blocked`), and `MAY` drop sounds that arrive faster than it finds reasonable.
- Capture needs a grant and shows the indicator (Section 8); a server can hide neither.
- Media rides the binding's TLS or QUIC encryption. There is no separate SRTP layer, and media never leaves the session's connection.

- A later session-recording extension must show its own indicator in the trusted zone.

§11 Definitions and checks

The definitions live in `ext/media-1/` of the bundle, and `tools/validate.py` runs 249 checks on them, and every check in the bundle passes.

File	What it holds
`vtx-media-1.cddl`	Socket additions (clip kinds, limits, permissions), capability values, OPEN params and the seven frame payloads
`vtx-ops-media-1.schema.json`	Every v1 and Surfaces 1 op, plus the media node, the pixel surface and `sound`
`vtx-app-media-1.schema.json`, `vtx-app-events-media-1.schema.json`	The application lines, both directions
`examples/valid/`	A scene; two wire traces, a remote desktop over QUIC and a buffered briefing over TLS; two application sessions, a voice agent and a briefing with a camera relay
`examples/invalid/`	19 schema and 10 scene-rule violations, 41 broken traces and 35 broken application lines, each labelled with the error it must raise
`examples/resources/`	`chime.opus` (Ogg Opus), `desk-av1.ivf` and `desk-h264.264`, encoded with ffmpeg
`examples/vectors.json`	The clip's hash and duration, an Opus packet as one datagram, an AV1 key frame in eight datagrams, and the two codec strings
`tools/gen_ext_media.py`, `tools/make_media.py`	Generate the schemas; rebuild the vectors, and with `--assets` the media files

The 249 checks:

- 34: the schemas are valid, the CDDL parses, and the scene example passes its schema and the scene rules;
- 9: the v1 and Surfaces 1 schemas reject media ops and `keys: "all"`;
- 29: each invalid scene line fails its own op's schema or a scene rule;
- 8: the resources: Ogg page CRCs, the Opus headers, hash, size and duration, AV1 temporal units, H.264 access units and both codec strings;
- 4: the vectors recompute, reassemble within 1,200 bytes and parse back;
- 38: the CDDL accepts 21 samples and rejects 15, and v1's CDDL still rejects Media 1 payloads;
- 43: a channel simulator replays both traces and catches every broken one for its labelled reason: channel IDs and opener, OPEN_OK first, codecs within capabilities, permissions and `once` grants, CONFIG before MEDIA, key frames, frame numbers and parts, datagram use and size, live MEDIA kept off the stream where datagrams exist, MARK on buffered channels only, who sends what, muted capture, media limits, and v1 windows on TLS;
- 84: every application line passes its schema, both sessions pass the application rules, and every broken line or session fails for its labelled reason.

Two more pycddl workarounds. pycddl matches a `.regexp` anywhere in a string, though RFC 8610's XSD regexps match the whole string, and it accepts a map that mixes the keys of two alternatives in a choice of maps. The validator now anchors every regexp and tries each map alternative on its own. Both fixes apply to v1 as well; its 444 checks still pass under the stricter reading.

§12 Wire summary

Media 1 fills v1's extension points and adds entries to registries that take new entries by specification; it changes no v1 rule.

Registry or extension point	Media 1 adds
Dynamic channel types	`audio`, `video`
Frame types on those channels	0x10 CONFIG, 0x11 MEDIA, 0x12 FEEDBACK, 0x13 MARK, 0x14 CUE, 0x15 STATE, 0x16 VIEW
Capability keys	`audio`, `audio.capture`, `audio.clips`, `video`, `video.capture`, `video.clips`, `surface.pixel`, `input.keylock`
Limits (`$$limit-ext`)	`media_kbps`, `media_channels`, `video_px`
Permissions (`$permission-ext`)	`microphone`, `camera`, `screen`, `keyboard.lock`
Resource kinds (`$res-kind-ext`)	`audio/ogg`, `audio/webm`, `video/webm`, `video/mp4`
Close codes	0x0E `user_stopped`, from the range v1 reserves for later specifications
Node types and surface kinds	The media node; surface kind `pixel`; `keys: "all"` on grids and pixel surfaces
Scene ops	`sound`
Intents on media nodes	`media.play`, `media.pause`, `media.seek`, `media.ended`
Application interface kinds	From the application: `stream`, `media`, `cue`, `mark`, `stream_close`. To it: `media`, `stream_state`, `stream_feedback`
Pointer fields	None new: pixel surfaces reuse Surfaces 1's `sx` and `sy`