VTX Viewers 1: shared sessions and recording

Nov 15, 2026 · VTX 1 · extension · draft d01 · MIT licence · vtx.foundation/spec/v1/viewers-1/

§1 Status and scope

Viewers 1 is an extension of VTX 1, draft d01, in conformance profile 1, built on the core at draft d03. In the core, a session has one active connection: a resume from a second device takes it over. Viewers 1 lets several clients attach to one session at once, each as a viewer with a role. That gives four things:

- **Shared sessions.** An owner invites others to watch, work along or approve. Pair programming with an agent, a support engineer looking over a customer's shoulder, and an incident room all use this.
- **Control of input.** One viewer at a time types into grids and remote desktops; control passes by asking and giving. Clicks and form submissions in components can come from anyone allowed to interact.
- **Approvals by several people.** An approval goes to every viewer who may approve. It can require a quorum, such as two different people for a production deploy.
- **Recording and calls.** The owner, a server policy or the application can record the session into a sealed, replayable file, with every viewer told so. Viewers can talk to each other through the server, or peer to peer over WebRTC.

Out of scope.

- Several sessions on one screen. Clients may already show several sessions side by side.
- Live streaming to large audiences. Viewers 1 aims at tens of viewers, not thousands.
- Editing a recording, and how long a server keeps one. Those are deployment policy.

What changes in v1. Nothing changes for a peer that does not offer the extension. A v1 resume still takes a session over; joining as a viewer is a separate request. Viewers 1 adds:

- a dynamic channel type, `viewers`;
- a capability key, `viewers`;
- a control frame for the waiting room, 0x22 WAIT;
- two close codes, 0x10 `not_admitted` and 0x11 `removed`;
- a permission, `recording`;
- a recording format, a CBOR sequence.

To express these, the v1 CDDL gains three sockets that v1 itself leaves empty: `$$hello-ext` for the join request, `$$ready-ext` for the viewer's role, and `$$approval-ext` for the quorum.

§2 Capabilities and limits

One capability key, `viewers`, carries the client's abilities. The server answers with what its policy grants. Every flag defaults to false.

Flag	The client can	Needed for
`join`	Attach to a session someone else started	HELLO `join`
`manage`	Invite, admit, change roles and remove viewers, as owner	INVITE, ADMIT, SET
`record`	Start and stop recordings, as owner	RECORD
`call`	Take part in calls between viewers	CALL; Media 1 audio or video as well
`webrtc`	Carry a call peer to peer	SIGNAL, CALL with p2p

The server's answer may add `max`: the most viewers a session takes. With `viewers` accepted, the client may open one `viewers` channel per connection; Section 11 lists its frames. A client that never opens one can still join and watch: READY tells it its role, and the trusted zone shows when the session is recorded.

Limits. Viewers 1 adds no negotiated limits. Five fixed numbers apply:

- A viewer waits at most 300 seconds for the owner.
- An invitation lives between 1 minute and 7 days, for 1 to 100 uses.
- A quorum is 2 to 5 people.
- Presence goes out at most 20 times a second per viewer.
- A server refuses viewers beyond its `max` with RESULT `limit`, or closes the joiner with `not_admitted`.

§3 Roles and joining

Every viewer has one role. A role is a bundle of rights, and the server enforces it on every frame.

Role	Sees	Types and points (with control)	Clicks and submits	Answers approvals	Manages viewers, records
`owner`	Everything	Yes	Yes	Yes	Yes
`operate`	All but owner-only nodes	Yes	Yes	Yes	No
`interact`	Nodes for interacting viewers	Yes	Yes	No	No
`approve`	Nodes for approvers	No	No	Yes	No
`watch`	Nodes for watchers	No	No	No	No

The user who started the session is its owner, viewer 1. Ownership does not move. Guests, who join without an account, get at most `interact`: approving and operating need an identity someone can be held to.

Ways in. A client joins with `join` in HELLO: the session ID, an invitation token, and for a guest a name. A v1 `resume` still takes a session over; `join` adds a viewer.

- **The owner's own devices** join without an invitation, as owner. A phone can follow a session the laptop runs.
- **Invitations.** The owner sends INVITE with:
  - a role;

    - optionally one account that may use it, or permission for guests;
    - a number of uses and a lifetime;
    - `admit`: `auto` or `knock`.

  INVITED returns the token and a link, `vtx://host/#join=<session>.<token>`. Both parts sit in the fragment, which never leaves the client. The client reads them and sends them in HELLO.

- **Knocking.** With `admit: "knock"`, the joiner signs in, then gets WAIT and stays in the waiting room for up to 300 seconds. The owner's client shows KNOCK in its trusted zone: name, account or “guest”, the role, where they connect from. The owner answers ADMIT, and may lower the role but never raise it above the invitation's. A refusal, or the wait running out, closes the joiner with 0x10 `not_admitted`.

  Diagram: joining by invitation with a knock · 3 parties, 10 messages. Drawn in the PDF version, `08-viewers-1.pdf`.

The owner decides in his own trusted zone; Eva's client learns her role from READY.

After joining.

- READY carries `viewer`, `role` and `recording`. Each viewer has its own state numbers: READY says state 0, and a reset snapshot at state 1 follows, holding what this role may see.
- Every client with a `viewers` channel gets ROSTER after every change: who is attached, their roles, who holds control, who is asking, and whether the session is recorded.
- The owner changes a role or removes a viewer with SET. A removed viewer's connection closes with 0x11 `removed` and does not resume.
- When the owner leaves, the others stay. Approvals still reach approvers; knocks wait until the owner returns or the wait expires.

Other extensions. A viewer's rights come from two things: its own account, and its role in this session. Joining someone's session never lends that person's rights.

Feature	With several viewers
Exec 1: exec and forwarding	Only connections signed in as the session's user, in role `owner`
Transfer 1	A download goes to the viewer the application names, by default the one holding control. Uploads come from viewers who may click
Clipboard, haptics	The viewer named in the line, by default the one holding control
Media 1 playback	A play stream goes to each viewer who can see its media node, on that viewer's own channel. Keys to a pixel surface come only from the viewer holding control
Media 1 capture	Each viewer's own microphone and camera, behind its own permission
Devices 1	Notices go to the owner. Approvals wake every approver's devices
Sensors 1	Read from the viewer the application names, by default the one holding control, behind that viewer's own permission
Screen readers (Sensors 1)	Every A11Y action but `focus` counts as an intent, from any viewer who may click. `focus` moves the application's focus only from the viewer holding control; other viewers keep their focus in their own client

Feature	With several viewers
Permissions	Asked of the viewer they concern, never of everyone

§4 Input, control and presence

Two people typing into one terminal at once produce garbage, but two people clicking buttons in one dashboard is fine. So Viewers 1 treats the two kinds of input differently.

Input	Frames	Who may send it
Control input	KEY, COMPOSE, POINTER, WHEEL, GESTURE, GAMEPAD, RESIZE, and PASTE into grids and pixel surfaces	Only the viewer holding control
Component input	INTENT, and PASTE into fields	Any viewer whose role may click: `owner`, `operate`, `interact`

A client does not send input its viewer may not send, and the server drops any that arrives. Every input event reaches the application with `viewer`, and `seq` counts per viewer, so v1's delivery guarantees hold for each viewer separately.

Control. ROSTER names the viewer holding control; when the session starts, that is the owner. CONTROL frames move it:

- `request`: ask for it. The viewer joins `asks`, which every viewer sees, oldest first.
- `give` with `to`: the holder or the owner hands it to a viewer who may type.
- `take`: the owner takes it back at any time.
- `release`: the holder gives it up, and it returns to the owner.

When the holder leaves, or is moved to a role that may not type, control returns to the owner, or to nobody while the owner is away.

The owner can switch `handover` from `ask` to `free` with SET. In free mode, any viewer who may type takes control by typing, once the holder has been idle for 2 seconds. That suits two people pairing closely; `ask` suits a support session.

Sizes.

- Each viewer sends its own DISPLAY, and each client lays out components for its own window.
- Grids sized to fill, such as a terminal, and pixel surfaces have one size: the holder's. Only the holder's RESIZE takes effect.
- Other viewers scale or scroll the grid to fit their window. The application's `display` event follows the holder.

Presence. A viewer sends PRESENCE with the node it points at or works in, and `at`, a position in that node's own units: cells in a grid, units on a surface. The server relays it to the other viewers with the sender's `viewer`, as datagrams on QUIC, at most 20 times a second.

- Clients draw other viewers' pointers, or a highlight on the node, with the viewer's name.

- Positions inside components are left out on purpose: each client lays components out differently, so only the node is shared.
- A client may follow another viewer, keeping that viewer's presence target in view.

§5 Approvals with several viewers

A v1 approval goes to the one attached client. With several viewers, it goes to every viewer whose role may approve: `owner`, `operate` and `approve`. Watchers, interacting viewers and guests never see it.

One answer by default.

- The first valid answer decides. Each answer is checked against the APPROVAL its viewer received, with the v1 digest and, for `uv`, that viewer's own passkey or card.
- The server then sends CANCEL to every other approver, so a prompt never lingers on a screen after someone else answered.
- `zone_result` tells the application who answered: `viewer`, and the account in `votes`.

Quorum. An application can ask for several people with `quorum` (2 to 5) on its `approval` line, and the zone APPROVAL carries it. This is the four-eyes rule, for a production deploy or a key rotation.

- Allows count once per account. The owner allowing from a laptop and a phone is one vote, not two.
- One deny decides at once.
- Approvers with a `viewers` channel get VOTE after each answer: who answered, how, and the count so far.
- When the count reaches the quorum, the server sends CANCEL to approvers who have not answered, and `zone_result` carries every vote in order, each with its proof.
- If too few approvers are attached, the approval waits. With Devices 1, the waiting approval wakes every approver's registered devices, not just the owner's.

Expiry. `expires` runs once for the approval, not per viewer. When it runs out, every approver gets CANCEL and the application gets `expired`.

§6 Session recording

A recording replays the session as a viewer saw it: every update, resource and event, in time. It does not capture pixels: it is the scene stream itself, so it stays small, stays sharp at any size, and its text can be searched. Privileged-access audits, agent runs and support cases are what it is for.

Who starts it.

- The owner, with RECORD;
- the application, with a `record` line;
- or a server policy, from the first moment of the session. Policy recording is how a bastion records every admin session.

No other viewer can start or stop one.

Everyone knows.

- While a session is recorded, every client shows it in its trusted zone, where content cannot hide or imitate the indicator.
- READY tells a joiner before the first update arrives, and the waiting room tells a knocking viewer before the owner admits them.
- ROSTER carries `recording` for clients with a `viewers` channel.

What goes in. A recording is made as a `watch` viewer sees the session. Owner-only and approver-only nodes stay out, so a recording can be shared with people who could have watched.

- In: the scene updates; the resources they use, once each; play streams, as `config` and `media` records; and events: joins, leaves, role and control changes, approvals and their answers, chapters.
- Out: input of any kind. Keys and form values can hold passwords. An application that wants an action on record says so with a `chapter` line.
- Call audio: only with `audio: true`, and only from viewers who allowed it. Each viewer in the call is asked the `recording` permission in its own trusted zone; those who decline are left out of the file.

The format. A recording is a CBOR sequence (RFC 8742), with the extension `.vtxr`:

1. A header: the session, host, application, start time, `audience: "watch"`, a reference display, and the server's signing key with its algorithm.
2. Records of the form `[t, kind, body]`, `t` being milliseconds since the start:
    - `update`: a v1 UPDATE payload;
    - `res`: a resource, before its first use;
    - `config`: a Media 1 CONFIG for a recorded stream, either a play stream or `call:<viewer>` for a speaker who allowed it;
    - `media`: one whole frame of that stream, reassembled, with its `pts` and key flag;
    - `event`: what happened;
    - `seal`.
3. A seal holds the SHA-256 of every byte before it, signed over `"vtx1-recording"` ‖ hash.

The server seals at least once a minute and at the end. A file cut short is still verifiable up to its last seal, and one changed byte breaks every later seal. State numbers count from a reset snapshot at state 1, and the server repeats a reset snapshot every five minutes so a player can seek.

Replay. A player is a VTX client without input. It applies the updates in time, shows events on a timeline, and offers chapters as jump points. Because it renders the scene itself, a recording made on a laptop replays readably on a phone. The server keeps recordings as its policy says. The owner can download one with Transfer 1, and the application hears where it was stored.

§7 Calls between viewers

People who share a session usually want to talk while they work. Viewers 1 builds calls from Media 1 rather than adding a media stack of its own. The server relays by default, and WebRTC carries the call peer to peer when every member can.

Relayed through the server. This is the default, and it works wherever the session works.

1. A viewer sends CALL `join` with `audio` and, if it likes, `video`.
2. Its client asks the Media 1 `microphone` (or `camera`) permission in its trusted zone.
3. The server opens a capture channel on that viewer's connection, with stream `call:<viewer>`.
4. For every other member, the server opens a play channel carrying that stream, titled with the speaker's name.

Everything else is Media 1: Opus, live on QUIC datagrams, the stream fallback on TCP, FEEDBACK for congestion, `muted` in STATE. A viewer who is not in the call receives no call streams. A relayed call can be recorded, from members who allow it (Section 6).

Peer to peer. When every member's client has `webrtc`, they may move the call to WebRTC:

- SIGNAL frames carry the SDP offer and answer and the ICE candidates through the `viewers` channel. The server relays each one to the named viewer, stamped with the sender.
- Once media flows directly, members send CALL `join` with `p2p`, and the server closes the relayed channels.
- DTLS-SRTP encrypts end to end, so the server never sees the media. A peer-to-peer call is therefore never recorded, and ROSTER shows `p2p` so everyone knows it is off the record.
- If the peers cannot reach each other, they stay on the relay.

Leaving. CALL `leave` ends a viewer's part: the server closes its channels and the others' play channels for its stream. Leaving the session leaves the call. ROSTER shows each member's `call` state: `audio`, `video`, `muted` and `p2p`.

§8 Application interface

An application keeps writing one scene, as in v1; the server fans it out to the viewers. It learns who is there, who did what, and can keep some nodes from some roles.

Per-role content. Any `open` op may carry `audience`: the roles that receive the node and its subtree.

```
{"op":"open","id":"keys","type":"region","title":"Signing keys","audience":["owner"]}
```

- The server filters: a hidden node, its children, and every op that touches them never reach that viewer. Neither do recordings, which see the session as `watch`.
- When a viewer's role changes, the server opens what the new role may see and closes what it may not.
- `audience` hides content; it is not access control. Actions an approver or owner must take still go through `approval`, and the application checks the `viewer` on every intent it acts on.

Lines from the application (`vtx-app-viewers-1.schema.json`):

Kind	Change
`approval`	New field `quorum`: allows needed from different users (2 to 5)

Kind	Change
`record`	New: `action` `start` or `stop`, optional `audio`
`chapter`	New: `label`, a named point in the recording
`clipboard_read`, `clipboard_write`, `haptic`, `permission`, capture `stream`, `download`	New field `viewer`: whose device; by default the one holding control

Lines to the application (`vtx-app-events-viewers-1.schema.json`):

Kind	Change
`viewer`	New: `joined`, `left`, `role` or `control`, with the viewer's ID, name, account (absent for a guest) and role
`recording`	New: `started` or `stopped`, by whom, and where the file went
`intent`, `input`, `clipboard_data`, `upload`, `upload_chunk`, capture `media`, `notice`	New field `viewer`: who it came from
`zone_result`	New fields `viewer` (who decided) and `votes` (every answer of a quorum approval, with its proof)

An agent that offers a deploy button to everyone, but deploys only with two approvals, does this:

```
{"kind":"approval","id":"deploy-41","title":"Deploy release 41 to
production","risk":"high","requester":{"name":"Release bot","kind":"agent"},"quorum":2}
```

It then waits for `zone_result`, whose `votes` name both people, ready to store with the release.

§9 Security and privacy

Invitation links leak. Links end up in chat logs, tickets and screenshots, so an invitation should assume it will be seen by the wrong person.

- Use `knock` for anything sensitive. A leaked link then only lets someone ask, and the owner sees the account, or “guest”, before saying yes.
- Name the account for `auto` invitations. A token alone is a bearer secret.
- Keep invitations short-lived and single-use; the defaults are one use and a lifetime the owner picks.
- The token sits in the link's fragment, which browsers and clients never send to a server, so it does not land in web server logs on the way.

No borrowed rights. A viewer acts with its own account and its role in this session, never with the owner's rights. Exec, forwarding and device management stay with the owner's own connections. Guests can never approve.

Impersonation.

- The trusted zone shows each viewer as the server verified them: the account, or “guest” with the name they typed, and the guest name is marked as claimed.

- KNOCK prompts appear only in the owner's trusted zone, so scene content cannot fake one to trick the owner.
- Presence and pointers are labelled with roster names, which the application cannot set.

Content the role should not see. `audience` keeps nodes from roles, and the server enforces it before anything leaves. But the scene is not the only channel: an application that writes a secret into a shared grid has shared it. Owner-only material belongs in owner-only nodes.

Recording consent.

- Every viewer is told before it sees anything: in READY, in the waiting room, and continuously in the trusted zone.
- Microphones are recorded only with each speaker's own permission.
- Input is never recorded.
- A viewer who does not consent can leave; nothing it did before joining was recorded on its behalf.
- Laws on recording conversations differ by country. The protocol makes recording visible and consent explicit, but whether a recording is lawful stays the operator's responsibility.

Recordings at rest.

- Seals make tampering evident, not impossible to attempt: the file still needs access control.
- Servers should encrypt stored recordings, keep them only as long as policy says, and log who downloads them.
- A recording shows what a watcher saw. That limits what it exposes, and also what it proves.

Peer-to-peer calls. WebRTC reveals each member's IP addresses to the others through ICE candidates. Clients should offer `p2p` only between signed-in accounts, or use relay-only candidates when the user prefers privacy. The relayed call reveals nothing beyond the server.

§10 Definitions and checks

The machine-readable definitions are in the bundle under `ext/viewers-1/`:

File	What it holds
`vtx-viewers-1.cddl`	The join request and READY fields, the quorum, WAIT, the capability, every frame of the `viewers` channel, and the recording format
`vtx-ops-viewers-1.schema.json`, `vtx-app-viewers-1.schema.json`, `vtx-app-events-viewers-1.schema.json`	`audience` on every open op, and the application lines and events, generated by `tools/gen_ext_viewers.py`
`examples/valid/trace-*.jsonl`	Five traces: an invitation with a knock, control, a demotion and removal; a recorded deploy with a quorum, a promotion and a guest; the owner allowing from two devices; `free` handover; a call relayed, recorded with consent, and then peer to peer
`examples/valid/recording.vtxr`	The recorded deploy, as a server writes it, with two signed seals
`examples/vectors.json`	The invitation link, the recording's hash and key, and the one-vote-per-account rule

File	What it holds
`examples/invalid/*.jsonl`	40 broken traces and 10 broken application lines, each naming the check that must catch it

`tools/make_viewers.py` writes the traces, the recording and the vectors. Every key derives from a fixed label and every signature is deterministic, so a rerun produces the same bytes. The v1 CDDL gains the three empty sockets of Section 1, and v1 alone still rejects every Viewers 1 form.

141 checks. Section 15 of `tools/validate.py` adds 141 checks, all passing, for 2,543 in the bundle:

- 9: the schemas and the CDDL load. v1 alone rejects the join, the viewer fields and the quorum, which the extension accepts. Earlier schemas reject `audience`.
- 55: 26 well-formed and 29 malformed samples against the CDDL, the recording's `config` and `media` records among them.
- 8: the vectors, and the recording. The recording's header and records pass the CDDL, its updates replay through the v1 scene rules with nothing beyond a watcher's view, both seals cover the bytes before them, and their signatures verify. One changed byte breaks the seal.
- 45: five valid traces replayed, and 40 invalid ones, each failing for its stated reason.
- 24: application lines and events against their schemas.

What the trace checks prove. The simulator follows the server, every connection and every viewer, and verifies each passkey sign-in:

- **Joining.** A join needs a live invitation, from the right account, or the owner's own account. A knock gets READY only after the owner allowed it. Roles never exceed the invitation, guests stay at `interact` or below, and viewer IDs count up.
- **What each viewer receives.** A joiner's state starts at 0 with a reset snapshot. No update carries a node hidden from the viewer's role. After a role change the server removes what the new role may not see and opens what it now may, before anything else. READY and WAIT say whether the session is recorded.
- **The roster.** Every ROSTER matches the session: viewers, roles, control, asks, call states and recording.
- **Input.** Control input comes only from the holder, and intents only from roles that may click. Only the holder or the owner gives control, only the owner takes it, and control returns to the owner when its holder is demoted. Only the owner switches to `free` handover; then control passes to another viewer only after the holder has been idle for 2 seconds.
- **Owner-only actions.** Invitations, admissions, role changes, removals and recording are honoured only from the owner. A removed viewer is closed with `removed` before anything else reaches it.
- **Approvals.** They go only to approvers, with the application's quorum. VOTE counts once per account, nothing is cancelled before it is decided, and every unanswered approver gets CANCEL after.
- **Calls.** A microphone channel opens only after the permission; call streams go only to members; signalling is relayed only between members with WebRTC, stamped with the true sender. Call audio reaches a recording only when it was started with audio, and only from speakers who allowed it.

§11 Wire summary

The `viewers` channel.

- Only the client opens it, with an odd channel ID and no OPEN params; at most one per connection.
- The server sends ROSTER at once, and again after every change.
- INVITE, SET and RECORD carry `req` and are answered once: INVITE by INVITED or RESULT, the others by RESULT.

Type	Name	Direction	Payload
0x10	ROSTER	S→C	`{ you, viewers, ? control, ? asks, handover, ? recording }`
0x11	PRESENCE	Both	`{ ? viewer, target, ? at }`; stream or datagram
0x12	CONTROL	C→S	`{ action, ? to }`: `request`, `release`, `give`, `take`
0x13	INVITE	C→S	`{ req, role, ? user, ? guests, ? uses, expires, admit }`, owner only
0x14	INVITED	S→C	`{ req, token, link, expires }`
0x15	KNOCK	S→C	`{ id, name, ? user, role, ? from, ? platform, expires }`, to the owner, shown in the trusted zone
0x16	ADMIT	C→S	`{ id, decision, ? role }`, owner only
0x17	SET	C→S	`{ req, ? viewer, ? role, ? remove, ? handover }`, owner only
0x18	RESULT	S→C	`{ req, ok, ? reason, ? message }`; reasons `role`, `not-found`, `policy`, `limit`, `unsupported`
0x19	VOTE	S→C	`{ id, viewer, decision, count, quorum }`, to approvers
0x1A	RECORD	C→S	`{ req, action, ? audio }`, owner only
0x1B	CALL	C→S	`{ action, ? audio, ? video, ? p2p }`: `join`, `leave`, `mute`, `unmute`
0x1C	SIGNAL	Both	`{ viewer, ? type, ? sdp, ? candidate, ? mid }`: WebRTC signalling, relayed

Additions to v1.

Where	Addition
HELLO	`join`: `{ session, ? invite, ? name, ? role }`
READY	`viewer`, `role`, `recording`
Control frames	0x22 WAIT, S→C, between AUTH and READY: `{ expires, ? owner, ? recording }`
APPROVAL	`quorum`: 2 to 5
Close codes	0x10 `not_admitted`, 0x11 `removed`; neither resumes
Capability keys	`viewers`
Dynamic channel types	`viewers`
Permissions	`recording`
OPEN_REFUSED reasons	`policy`, `in-use`, `role`
Scene ops	`audience` on `open`

Where	Addition
Application interface kinds	`record`, `chapter`, `viewer`, `recording`; new fields on approval, targeted lines and events from viewers
CDDL sockets	`$$hello-ext`, `$$ready-ext`, `$$approval-ext`

Outside the VTX connection.

- The invitation link: `vtx://host/#join=<session>.<token>`.
- Recordings: CBOR sequences, `.vtxr`, signed over `"vtx1-recording"` ‖ hash. A vendor media type, `application/vnd.vtx.recording+cbor-seq`, joins the IANA requests when the extensions freeze; the core's requests do not need it.