VTX Sensors 1: sensors, gamepads and surface accessibility

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

§1 Status and scope

Sensors 1 is an extension of VTX 1, draft d01, in conformance profile 2, built on the core at draft d03. It completes the input side of VTX in three places that the core leaves to extensions:

- **Device sensors.** An application can read the user's motion sensors, location, ambient light, proximity and air pressure, each behind its own permission in the trusted zone. Games steer by tilting, field apps find the nearest depot, and dashboards dim in a dark room, all without native code.
- **Gamepads as datagrams.** The core lets an extension carry GAMEPAD as QUIC datagrams, where the newest state wins. Sensors 1 defines how, so that stick movement never waits behind a lost packet and a button press is never lost.
- **Accessibility on surfaces.** v1 maps components to the platform's accessibility API, and sends accessibility actions to the server for grids. Sensors 1 extends both to surfaces:
  - vector items get roles, values and actions;
  - pixel surfaces, such as a remote desktop, get an accessibility overlay that the server fills from the remote system's own accessibility tree.

Already in v1. Keyboards, pointers, touch, pens, gestures, gamepads and joysticks on the stream, haptics, dictation, and accessibility actions on components and grids. Sensors 1 changes none of it.

Out of scope.

- XR poses and hand tracking.
- Raw camera frames (Media 1 covers capture).
- Bluetooth and USB devices.
- Health sensors such as heart rate.
- A full bridge of a remote desktop's accessibility tree with every property and event. The overlay covers what a screen reader needs to find and use controls. The full bridge stays on the list for a later extension.

What changes in v1. Nothing changes for a peer that does not offer the extension. Sensors 1 adds:

- a dynamic channel type, `sensor`;
- capability keys `sensors`, `input.datagrams` and `surface.a11y`;
- permissions `motion`, `location` and `environment`, the first two reserved by v1, plus `location.precise`;
- accessibility actions `focus` and `set`;
- a limit, `sensor_hz`.

To carry the new actions and the item they act on, the v1 CDDL gains two sockets that v1 itself leaves empty: `$in-a11y-ext` and `$a11y-action-ext`.

§2 Capabilities and limits

Key	Client offers	Server answers
`sensors`	A map from each sensor it has to `max_hz` (location: `max_per_min`, and `precise` when it can give precise fixes)	The sensors its policy lets applications use, with the same values
`input.datagrams`	`true`: it can send GAMEPAD as QUIC datagrams	`true` to accept them
`surface.a11y`	`vector` and `pixel`: which kinds of surface it can expose to assistive technology	What the server will send

The sensors are:

- motion: `accelerometer`, `gyroscope`, `magnetometer`, `orientation`, `gravity` and `linear-acceleration`;
- environment: `light`, `proximity` and `pressure`;
- `location`.

A client lists only what it actually has: a desktop typically offers `light` at most, a phone most of them.

Limits.

- `sensor_hz` (client limit, default 400) caps the samples per second over all open sensor channels together. A server that would exceed it opens fewer channels or asks for lower rates.
- A single channel asks for 1 to 1,000 samples per second, or 1 to 60 location fixes per minute, and never more than the client's `max_hz` or `max_per_min`.
- `batch` (1 to 1,000) lets a client gather several readings into one SAMPLE, which saves radio wake-ups on phones at the cost of latency.

§3 Sensors

Subscribing. One sensor per channel, and the server opens it:

1. The application asks with a `sensor` line: a name, the sensor, a rate, and for motion the axes to report in.
2. The server asks the sensor's permission in the client's trusted zone, unless a grant for this server is still in force.
3. With the grant, the server opens a `sensor` channel. OPEN_OK carries the rate the client will actually deliver, which may be lower than asked, never higher.
4. Readings flow as SAMPLE, or FIX for location, until either side closes the channel.

A refusal arrives as OPEN_REFUSED with `permission`, `unsupported` or `limit`. With several viewers (Viewers 1), the sensor is read from the viewer the application names; by default, the one holding control.

Sensors	Permission	What the zone says
Motion: accelerometer, gyroscope, magnetometer, orientation, gravity, linear acceleration	`motion`	The server reads how the device moves and turns

Sensors	Permission	What the zone says
Environment: light, proximity, pressure	`environment`	The server reads light, nearness and air pressure around the device
Location, coarse	`location`	The server learns the area the device is in, to about a kilometre
Location, precise	`location.precise`	The server learns where the device is, to a few metres

Permission scopes are v1's: `once`, `session` or `always`, stored per server identity and revocable in the client.

Readings. A SAMPLE carries `r`, a list of readings, oldest first. Each reading is `[t, values…]`, with `t` on the client's input clock in microseconds, the same clock as every v1 input event.

Sensor	Values	Unit
accelerometer, gravity, linear-acceleration	x, y, z	m/s²
gyroscope	x, y, z	rad/s
magnetometer	x, y, z	µT
orientation	x, y, z, w (a unit quaternion)	none
light	illuminance	lux
proximity	distance	cm
pressure	air pressure	hPa

Motion axes follow the W3C Generic Sensor conventions. With `frame: "device"`, the default, they are fixed to the device; with `frame: "screen"`, they follow the screen as the user holds it, which is what a game wants.

Location. FIX carries `lat`, `lon` and `acc` (95% radius, metres), and when known `alt`, `alt_acc`, `heading` and `speed`.

- Coarse fixes are rounded to 0.01 degree, and their accuracy is never better than 1,000 metres. The client does the rounding, so a precise position never leaves the device without the precise grant.
- Fixes come at most `per_min` times a minute and always on the stream: they are few, and each one matters.

Transport.

- On QUIC, SAMPLE may travel as datagrams. A lost reading is replaced by the next one, as with live Media 1 frames. A SAMPLE sent as a datagram fits max_datagram (default 1,200 bytes) as a whole frame; a batch that would not fit is split into several SAMPLE frames.
- Elsewhere, and for FIX and STATE, frames go on the stream.
- Readings never arrive faster than the delivered rate, with 10% leeway for clock jitter.

State. The client reports STATE when it stops delivering: `paused` (backgrounded, or the user paused it), `unavailable` (the hardware stopped or power saving), and `active` again. No readings arrive while a channel is paused.

§4 Gamepads over datagrams

A game or a remote desktop feels sluggish when a stick movement waits behind a lost packet. On one reliable stream, every later state waits until the lost one is resent: head-of-line blocking. QUIC datagrams avoid that, because they are never resent. But a button press that only travelled in a lost datagram would simply be gone. Sensors 1 therefore splits the two:

- **Datagrams** carry the gamepad's full state whenever an axis or analogue trigger changes, up to the device's polling rate. A lost one is replaced by the next.
- **The stream** carries the full state whenever a button or hat changes: every press and every release, each in its own frame. Only after that may datagrams carry the new button state.

Diagram: gamepad states on two paths · 5 states, 1 lost. Drawn in the PDF version, `09-sensors-1.pdf`.

The stick state at t2 is lost and simply replaced at t4; the press at t3 arrives because it went on the stream.

Newest wins. Every GAMEPAD frame is the device's full state, and `t` orders them. The server applies a state only if its `t` is newer than the last it applied for that device, whichever path it came by.

- A stream frame that arrives after a newer datagram is skipped. Nothing is lost by that: the datagram already holds the same button state, because datagrams only carry button states that went on the stream first.
- The application sees states in `t` order, at most one per device per state it reads. A server may coalesce states when the application reads slower than the pad moves.

Rules.

- GAMEPAD goes as a datagram only when both sides accepted `input.datagrams` and the connection has datagrams. Otherwise every state goes on the stream, as in v1.
- Every state carries every button and axis the device announced (v1 DEVICES), so a lost frame never leaves a value unknown.
- `t` rises strictly per device.
- Datagrams are not counted against the input channel's credit; the QUIC connection's own congestion control paces them.

The same split works for joysticks. Haptics stay on the stream from server to client, as v1 defines: a lost rumble is worse than a late one.

§5 Accessibility on surfaces

Components are accessible in v1 because the client renders them itself and maps them to the platform's accessibility API. Surfaces are drawings and video, so a screen reader sees one image. Sensors 1 gives both kinds of surface a semantic layer, and lets accessibility actions reach the items in it.

Vector items. Any vector item may carry accessibility props:

- `role`: one of `button`, `link`, `checkbox`, `switch`, `radio`, `slider`, `spinbutton`, `progressbar`, `textbox`, `combobox`, `menuitem`, `tab`, `listitem`, `treeitem`, `heading`, `image`, `text`, `group`, `list`, `dialog`, `window`, `toolbar`, `region`, `table`, `row` or `cell`.
- `value`, and for ranged roles `min`, `max` and `step`.
- `checked`, `expanded`, `selected`, and `level` for headings.
- `order` (reading and focus order), `focusable` and `description`.
- `actions`: the accessibility actions the item handles beyond `activate`, which an `intent` or `link` already implies.

The Surfaces 1 `label` is the accessible name; a text item's text serves when it has none. `edit` changes these props like any other, so a slider's value stays current.

Pixel surfaces. A remote desktop's pixels carry no structure, but the remote system has one: Windows UI Automation, macOS Accessibility, AT-SPI. The server sends it as an `a11y` op: a list of nodes, each with:

- an `id`, a `role` and a `label`;
- a `rect` in the surface's pixels;
- optionally a `parent`, which comes earlier in the list;
- and the same value, state and action props as vector items.

The list replaces the previous one; an empty list clears it. The client builds an accessibility tree from it over the video, so a screen reader can find the Send button and its label, and a switch user can reach it.

Rules the client and the server keep.

- An item with a role has an `id`, and an interactive role has an accessible name.
- Ranged roles carry numbers with `min` < `max` and `value` between them.
- `checked` is for checkboxes, switches, radios and menu items; `level` is for headings.
- `increment` and `decrement` are for sliders and spin buttons; `set` is for text boxes, combo boxes, sliders and spin buttons.

Actions. The v1 A11Y frame gains `item`, naming the vector item or overlay node, and two actions:

Action	Meaning
`focus`	The assistive technology moved focus to the item. A remote desktop server moves the remote focus too
`set`	Set the item's value to `value`: text for a text box, a number within range for a slider

The v1 actions apply to items as well: `activate`, `increment`, `decrement`, the four scrolls and `dismiss`. An action is valid only on an item that handles it. A11Y comes only from the client's accessibility device, which assistive technology drives, so the application can tell it apart from a mouse click.

Honesty. An overlay that disagrees with the pixels misleads exactly the users who cannot see the difference. A server sends an overlay only from the remote system's own accessibility tree, updates it when that tree changes, and clears it when it cannot keep it current.

§6 Application interface

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

Kind	Fields	Effect
`sensor`	`name`, `sensor`; `hz`, `frame` and `batch` for motion and environment; `per_min` and `accuracy` for location; `viewer` (Viewers 1)	Subscribe: the server asks the permission, then opens the channel
`sensor_close`	`name`	Stop a subscription
Scene ops	Accessibility props on vector items; the `a11y` op for pixel surfaces	As in Section 5

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

Kind	Fields	Meaning
`sensor`	`name`, `r`	Readings, `[t, values…]` as in Section 3
`fix`	`name`, `t`, `lat`, `lon`, `acc`, and `alt`, `alt_acc`, `heading`, `speed` when known	A location fix
`sensor_state`	`name`, `state`, `reason`, `hz`	`open` with the delivered rate, `active`, `paused`, `unavailable`, `refused` with the reason, `closed`
`input`	`event: "gamepad"` or `"a11y"`, `data`	As in v1. Accessibility actions now carry `item` and, for `set`, `value`

The server coalesces gamepad states to the application's pace and delivers them in `t` order, whichever path they took. It also checks every accessibility action against the item's role and actions before passing it on. An application asks for a sensor like this:

```
{"kind":"sensor","name":"tilt","sensor":"orientation","hz":60,"frame":"screen","batch":4}
```

Then it waits for `sensor_state` `open`, or `refused`, before counting on readings. A refusal is normal: the user said no, or the device has no such sensor. The application should offer another way, such as on-screen buttons instead of tilt.

§7 Security and privacy

Sensors reveal more than they seem to. Published research shows each of these leaks:

- Accelerometer and gyroscope readings can reveal what someone types on a touch screen, and gyroscopes pick up nearby speech.
- Magnetometers and calibration errors fingerprint a device.
- Ambient light has leaked screen contents.
- Location is location.

Sensors 1 answers with:

- **A permission for everything.** Motion, environment and location each have their own permission, asked in the trusted zone. Even light needs one, though browsers once read it freely.

- **Rate caps.** No channel exceeds the client's `max_hz`, and all channels together stay under `sensor_hz`. Clients should offer lower maxima than their hardware can do: 60 Hz steers a game, and few applications need more. A client may lower precision too, rounding readings to what the purpose needs.
- **Visible use.** While a sensor channel is open, the client shows it in its trusted zone, as with a microphone: a motion or location indicator the user can tap to stop it.
- **Focus only.** Clients pause motion channels while the session is not in front, and say so with STATE `paused`. Keystroke inference needs the moments when the user types elsewhere.
- **Coarse by default.** Coarse location is rounded on the device, and precise needs its own grant, which the user can refuse while keeping coarse.

Viewers. Sensors belong to one viewer's device: the one the application names, by default the one holding control. Another viewer's sensors are never read without that viewer's own permission. Readings go to the application, not to other viewers. With screen readers on several viewers, every A11Y action but focus counts as an intent, accepted from any viewer whose role may click. Focus moves the application's focus only when it comes from the viewer holding control; any other viewer's screen reader keeps its focus in its own client and does not send it.

Gamepad datagrams. Datagrams ride inside the QUIC connection, so they get the same encryption and authentication as everything else. They are not resent, which is the point, and they are not counted against input credit. A client must still not flood them: it sends at most one per polling interval of the device. A server may drop a client's datagrams beyond 1,000 a second and close the connection with `limit_exceeded`.

Accessibility overlays. An overlay changes what a screen reader announces, so a dishonest one could steer a blind user to the wrong button. Two rules keep this in check:

- Overlays come only from the server's own reading of the remote system's accessibility tree, never from application content.
- They never cover the trusted zone, which the client draws itself.

The client also announces the overlay as part of the remote desktop, so users know whose labels they hear. Accessibility actions reach the server as input from the accessibility device, so an application can treat them as user actions with the same `seen` checks as any click.

§8 Definitions and checks

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

File	What it holds
`vtx-sensors-1.cddl`	The permissions, the A11Y additions, the `sensor_hz` limit, the capabilities, OPEN params and every frame of the `sensor` channel
`vtx-ops-sensors-1.schema.json`, `vtx-app-sensors-1.schema.json`, `vtx-app-events-sensors-1.schema.json`	Accessibility props on vector items, the `a11y` op, and the application lines and events, generated by `tools/gen_ext_sensors.py`

File	What it holds
`examples/valid/trace-*.jsonl`	Three traces: orientation as datagrams with a gamepad over QUIC; coarse location, a refused precise request and ambient light over TLS; a screen reader working a vector mixer and a remote desktop's overlay
`examples/valid/a11y-scene.jsonl`	The mixer and the remote desktop with its overlay
`examples/invalid/*.jsonl`	36 broken traces, 7 accessibility ops that fail the schema, 11 that break the rules, and 8 broken application lines

`tools/make_sensors.py` writes the traces and examples; the readings are computed, so a rerun gives the same bytes. The v1 CDDL gains the two empty sockets of Section 1.

114 checks. Section 16 of `tools/validate.py` adds 114 checks, all passing, for 2,543 in the bundle:

- 6: the schemas and the CDDL load. v1 alone rejects the new A11Y fields and the new permissions, which the extension accepts.
- 28: 12 well-formed and 16 malformed samples against the CDDL.
- 23: the accessibility scene passes the schema and the rules, earlier schemas reject item roles, and every broken op fails for its stated reason.
- 39: three valid traces replayed, and 36 invalid ones, each failing for its stated reason.
- 18: application lines and events against their schemas.

What the trace checks prove.

- **Sensor channels.** Only the server opens one, only for a sensor the server's answer includes, and only after the user allowed the matching permission. Precise location needs its own grant.
- **Rates.** No rate exceeds what the client offers. OPEN_OK never promises more than was asked, and all channels together stay under `sensor_hz`.
- **Readings.** Each has the values its sensor defines, in rising time and no faster than the delivered rate. Batches stay within `batch`, nothing arrives while paused, and coarse fixes are really coarse. Fixes never travel as datagrams, and nothing does on a connection without them.
- **Gamepads.** Datagrams need `input.datagrams`. Every state carries every button and axis of an announced device, time rises per device, and no datagram carries a button state that has not gone on the stream.
- **Accessibility.** Actions come only from the accessibility device and only for granted kinds of surface. They name items that exist, and each action must suit its item: no `increment` on a switch, no `set` without a value or out of range, no focus on a heading.

Two more pycddl workarounds came with this extension, both in the README:

- pycddl accepts a map of only unknown keys for a map whose members are all optional;
- it does not enforce + on the repeated tail of an array.

`tools/validate.py` checks both itself, which also tightens the earlier extensions' capability maps; they all still pass.

§9 Wire summary

The `sensor` channel.

- Only the server opens it, with an even channel ID, after the sensor's permission.
- OPEN params: `{ name, sensor, ? hz, ? per_min, ? accuracy, ? frame, ? batch }`.
- OPEN_OK params: `{ ? hz, ? per_min }`, never more than asked.
- Refusal reasons: `permission`, `unsupported`, `limit`.

Type	Name	Direction	Payload
0x10	SAMPLE	C→S	`{ r }`: readings `[t, values…]`, oldest first; stream or datagram
0x11	FIX	C→S	`{ t, lat, lon, acc, ? alt, ? alt_acc, ? heading, ? speed }`; stream only
0x12	STATE	C→S	`{ state, ? reason }`: `active`, `paused`, `unavailable`

Additions to v1.

Where	Addition
GAMEPAD	May travel as a QUIC datagram with `input.datagrams`; button and hat changes always on the stream; newest `t` wins
A11Y	`item` and `value`; actions `focus` and `set`; items on vector and pixel surfaces
Capability keys	`sensors`, `input.datagrams`, `surface.a11y`
Limits	`sensor_hz` (default 400)
Permissions	`motion`, `location`, `location.precise`, `environment`
Dynamic channel types	`sensor`
Scene ops	Accessibility props on vector items; `a11y` on pixel surfaces
Application interface kinds	`sensor`, `sensor_close`, `fix`, `sensor_state`
CDDL sockets	`$$in-a11y-ext`, `$a11y-action-ext`

Sensors 1 needs no registrations outside the VTX registries.

With this extension, the first nine extensions of the roadmap are complete:

- Surfaces, Themes and Composites;
- Media, Transfer and Exec;
- Devices, Viewers and Sensors.

What remains is the consistency pass across all nine.