VTX Windows 1: window surfaces and seamless apps

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

Windows 1 lets a server show an application as the client's own windows, each with the device's frame, taskbar entry and window switching, instead of one picture of a whole remote desktop. It is the RemoteApp of VTX, and the first step toward a full desktop session that replaces RDP on the hosted desktop.

§1 Status and scope

Windows 1 is an extension of VTX 1, draft d01, in conformance profile 3, built on the core at draft d03. It changes no rule of the core. It adds:

- a `window` node type: a top-level window on the client that holds a Media 1 pixel surface (a remote application's window) or ordinary scene content (a VTX application's second window);
- roles for dialogs, tool palettes, menus and tooltips, with owners, modal dialogs, and popups placed against their owner;
- the WINDOW input frame, which tells the server what the user did to a window: focus, move, resize, minimize, maximize, full screen, close or dismiss;
- a `window` op, so an application that draws its own title bar can start the client's native move or resize;
- titles, application IDs for grouping in the taskbar, icons and taskbar progress;
- the `fullscreen` permission.

A remote application's windows need Media 1 for their pixels. A VTX application's windows need nothing else.

Not in Windows 1. Several monitors (layouts, scale per monitor, a window across two screens) are Displays 1, the next extension. Tray icons and global hotkeys stay out: the client's window manager decides what is on the device. Drive, printer, USB and smart-card redirection are separate extensions.

§2 Capabilities and limits

One capability, `surface.window`, under the `surface.` prefix of the surface kinds. The client lists what its window manager can do, and the server sends only what the client accepted.

Key	Value	Meaning
`max`	Number	Most windows open at once (default 16); the server never opens more, and falls back to the whole desktop when an application needs more (Section 8)
`popups`	`true`	Menus and tooltips as their own windows, placed against their owner
`frameless`	`true`	Windows with `frame: "none"`, whose content draws its own title bar and asks the client to move it
`icons`	`true`	Window icons from image resources (Surfaces 1)
`progress`	`true`	Progress on the window's taskbar entry

During the beta, both sides name the draft in `drafts`: `"windows-1": 1`.

The pixels of a remote window travel as a Media 1 video stream, so the client's `video` capability and `media_channels` limit bound how many remote windows can show at once. A static window costs little: a stream sends frames only when its window changes.

§3 Window nodes

A window is a scene node of type `window`, opened at the root like any other node, that the client shows as one of its own windows. Its content is its children: one Media 1 pixel surface for a remote application's window, or ordinary scene content for a VTX application's own windows. Ops change it as they change any node: `patch` for the title, size or state, `remove` to close it, and v1's `focus` and `attention`.

```
{"op": "open", "id": "w-mail", "type": "window", "title": "Inbox – Mail", "app":
"org.gnome.Evolution",
 "icon": "sha256:8b61…", "at": [120, 80], "size": [1100, 720], "min": [480, 320]}
{"op": "open", "id": "s-mail", "type": "surface", "kind": "pixel", "parent": "w-mail",
"stream": "win-mail",
 "alt": "Evolution, inbox", "size": [1100, 720], "fit": "none", "mouse": "move",
"keys": "raw"}
```

Prop	Meaning
`title`	Required. Shown in the frame and the taskbar, with the server's host beside it (Section 10)
`app`	The application's ID, such as `org.gnome.Evolution`, so the taskbar groups its windows
`icon`	An image resource for the frame and the taskbar, with `icons`
`role`	`normal` (default), `dialog`, `utility` (a tool palette that stays above its owner), `popup` (a menu) or `tooltip`
`owner`	The window this one belongs to: required for popups and tooltips, usual for dialogs and palettes
`modal`	A dialog that takes all input meant for its owner until it closes
`at`	Where the content should start, px on the display. The client may place the window elsewhere, and says so with WINDOW `move`
`offset`	Popups and tooltips only: where they start, px from the owner's content origin
`size`	Required. The content's size, px, without the client's frame
`min`, `max`, `resizable`	What the user may resize it to
`state`	`normal`, `maximized`, `minimized` or `fullscreen`
`frame`	`native` (default): the client draws its own frame. `none`: the content draws its own title bar, with `frameless`
`progress`	0 to 1, shown on the taskbar entry, with `progress`
`audience`	As on every open op (Viewers 1)

Rules.

- A window sits at the root, and stays there: it is never inside another node, and `move` never puts it anywhere else.
- `owner` names an open window other than itself. Popups and tooltips have an owner and an `offset`, never `at`. Only a dialog with an owner can be `modal`.
- `size` stays within `min` and `max`. `role`, `owner`, `modal`, `frame` and `app` never change after the window opens.
- A window that owns open windows cannot be removed until they are. The server closes a dialog before its owner.

A remote application's window. The hosted desktop's compositor opens one window node per application window, with one pixel surface inside it and one Media 1 play stream per window. The surface's pixels are the window's content only: the client's frame replaces the remote one, and `fit: "none"` keeps one remote pixel per device pixel at scale 1. When the remote window resizes, the server patches both the window's and the surface's `size`, and the stream's next CONFIG carries the new resolution.

A VTX application's windows. Any scene content can sit in a window: a second region, a log, a vector surface. An agent can show its plan in one window and a live log in another, and the user arranges them with the device's own window manager.

> **Diagram: seamless windows · 3 remote windows, 3 native windows, 1 way back.** Drawn in the PDF version, `10-windows-1.pdf`.

Every remote window travels as its own window node, pixel surface and stream, so the device frames, stacks and switches them like local windows. What the user does to a window comes back as WINDOW input, and the compositor applies it to the application.

§5 Moving, resizing and closing

The client's window manager owns the windows: the user moves, resizes, snaps and switches them with the device's own gestures. The client tells the server what changed with WINDOW, a new input frame on channel 2. Input keeps its order, so a focus change always arrives before the keys that follow it.

```
{"t": 1600000, "window": "w-compose", "action": "resize", "size": [900, 650], "seen": 6}
```

Action	Sent when	Fields	The server
`focus`	The window became the active one		Makes the remote window active
`blur`	It is no longer active, and no other window of the session took over		Deactivates it
`move`	The user moved it	`at`	Tells the remote side where it is, so menus open in the right place

Action	Sent when	Fields	The server
`resize`	The user resized it	`size`	Resizes the remote window, then patches the window's and the surface's `size`
`minimize`, `maximize`, `restore`, `fullscreen`	The user changed its state	`size` for the last three	Applies the state and patches `state`
`close`	The user clicked the close button		Asks the application to close. The window stays until the server removes it, so the application can ask to save first
`dismiss`	A popup or tooltip lost to a click elsewhere or Escape		Closes it

Rules.

- WINDOW names an open window. Popups and tooltips only get `dismiss`; other windows never do.
- `move` carries `at`, and `resize` carries `size`. A resize stays within `min` and `max`, and never touches a window with `resizable: false`.
- Between the user's resize and the server's patch, the client scales or crops the old pixels; it never waits for the server before following the user's hand.

Frameless windows. Many applications draw their own title bar: GNOME's header bars, Chrome, Electron applications. On the hosted desktop the compositor sees the press on that title bar and asks for an interactive move; the server passes this on with the `window` op:

```
{"op": "window", "id": "w-compose", "action": "begin_move"}
{"op": "window", "id": "w-compose", "action": "begin_resize", "edge": "se"}
```

The client starts the device's own move or resize, as if the user had grabbed its frame, and reports the result with WINDOW. The server sends the op only while a pointer button is held in that window, after the press the client sent; the client ignores it at any other time, so a server cannot drag windows around on its own. Popups and tooltips never move.

§6 Dialogs, menus and popups

A window can belong to another, its `owner`. The client keeps owned windows above their owner, moves them with it where the device does that, and minimizes them together.

Role	Placed by	Frame	What the user can do
`normal`	`at`, or the client	The device's, or its own with `frame: "none"`	Everything
`dialog`	Centred on its owner, unless `at` says otherwise	The device's	Move, close; resize unless `resizable: false`
`utility`	`at`, or beside its owner	A small one, where the device has it	Move, close; it stays above its owner but never takes focus from it for long

Role	Placed by	Frame	What the user can do
`popup`	`offset` from the owner's content origin	None	Choose an item; anything else dismisses it
`tooltip`	`offset` from the owner's content origin	None	Nothing: it never takes focus or input

Modal dialogs. A dialog with `modal: true` takes all input meant for its owner. While it is open, the client sends no keys or pointer presses to the owner's content, and activating the owner brings the dialog forward instead. The server closes the dialog before it removes the owner.

Popups. A menu is the hardest window to get right remotely, because it must appear exactly next to what was clicked. A popup gives its position against its owner, so the client can place it correctly whatever the device did to the owner, and keeps it on screen by flipping or sliding it like a local menu. A click outside it, or Escape, sends WINDOW `dismiss`, and the server removes it. A client without `popups` never gets popup windows: the server shows menus inside the owner's own pixels instead, as a whole-desktop session does.

§7 Focus, keys and attention

One window of the session is active at a time, or none, when the user is in another application on the device. The client reports every change with WINDOW `focus` and `blur`.

- **Keys go to the active window.** A KEY event targets a node inside the active window, and the client sends none to any other. Pointer events go to the window under the pointer, as on the device.
- **The server may move focus between its own windows**, with v1's `focus` op, but only while one of them is active: opening a dialog and focusing it is fine. When none is active, it asks with `attention` instead, and the client flashes the taskbar entry rather than stealing focus from another application.
- **Keyboard lock** (Media 1's `input.keylock`) works per window: the shortcuts the device would take are sent to the remote window only while that window is active and locked.
- **Full screen** follows the user: WINDOW `fullscreen` from the client, after a menu choice or F11 on the device. A server that wants a window full screen on its own, for a slide show, asks the `fullscreen` permission in the trusted zone first. The device's own way out of full screen always works.

§8 With other extensions, and without windows

Extension	With windows
Media 1	One play stream per remote window. A resize changes the stream's resolution in its next CONFIG, and input maps onto each window's pixel surface as on any pixel surface
Surfaces 1	Window icons are image resources. A VTX application's window can hold vector and image surfaces
Transfer 1	A file or folder dragged from the device onto a remote window arrives through the window's pixel surface as a drop target. Copy and paste of files works between windows and the device as on the whole desktop
Sensors 1	Each window's pixel surface carries its own accessibility overlay, so a screen reader sees each remote window apart, as it sees local ones

Extension	With windows
Viewers 1	Each viewer gets its own native windows, placed by its own device. WINDOW input that changes the remote window (resize, state, close) and server-started moves come only from the viewer holding control; another viewer's moves stay on its own device. `audience` hides whole windows from roles
Displays 1	Several displays: at is a position in the client's layout of displays, display places a window on one of them, and maximize and full screen carry the new position. A full desktop becomes one full-screen window per display

Without windows. A client without `surface.window` gets the session as before: the whole remote desktop as one Media 1 pixel surface, the remote window manager's frames included. A VTX application that opens windows shows their content inside the main window instead, each as a region. A server also falls back to the whole desktop when an application would open more windows than the client's `max`, rather than leave some out.

Hosted desktop and seamless apps together. One server can offer both: the full desktop for a user who wants it, and single applications (published applications) as seamless windows. Which one a session gets is the session's own choice, made when it starts; listing and launching published applications, and choosing the host they run on, is Brokering 1.

§9 Application interface

On a hosted desktop, the part that knows about windows is the compositor: a small VTX bridge inside it (or beside it, as an X11 or Wayland window manager) writes one line per change, and the server turns those lines into ops and Media 1 streams. A VTX application writes the same lines for its own windows.

The compositor opens a window, a pixel surface inside it and a play stream that feeds the surface:

```
{"op":"open","id":"w-mail","type":"window","title":"Inbox —
Mail","app":"org.gnome.Evolution","icon":"sha256:8b61…","at":[120,80],"size":
[1100,720],"min":[480,320]}
{"op":"open","id":"s-mail","type":"surface","kind":"pixel","parent":"w-
mail","stream":"win-mail","alt":"Evolution, inbox","size":
[1100,720],"fit":"none","mouse":"move","keys":"raw"}
{"kind":"stream","name":"win-
mail","dir":"play","media":"video","codec":"av1","w":1100,"h":720,"fps":30,"mode":"live"}
```

What the user does to the window comes back as an input event, which the compositor carries out on the real window before it answers:

```
{"kind":"input","event":"window","data":{"t":1200000,"window":"w-
mail","action":"resize","size":[1280,800],"seen":1}}
```

The compositor resizes the application's window, then patches both the window and its surface to the size the application accepted, which may differ from the one asked for (a terminal rounds to whole character cells). A `close` is passed to the application as its own close request; the window goes only when the application closes it and the compositor writes `remove`. A dialog that asks "Save changes?" first is therefore just another window.

The compositor maps the remote desktop's window types onto roles: X11 `_NET_WM_WINDOW_TYPE` and Wayland xdg-shell toplevels, xdg-popups and dialogs with a parent each have a direct counterpart. Windows the remote desktop draws for itself (its panel, its desktop background, its notifications) are not opened as windows: the client's own device has those already.

§10 Security

A remote window looks like a local one by design, which is also the risk: a server could draw a window that imitates the device's own password prompt or another application. The rules below keep a remote window recognisable and keep the server from taking over the device.

- **The host beside the title.** The client shows the server's host (the name the user connected to, as v1 shows it in the trusted zone) beside every window's title, in the frame and in the taskbar entry. With `frame: "none"`, the client still shows the host in the taskbar entry and in its window switcher, which the server cannot draw over.
- **No covering the device's own interface.** Windows are ordinary windows of the client: they cannot be placed over the device's taskbar, lock screen or system prompts, nor be made always-on-top. Popups and tooltips stay within reach of their owner.
- **No focus stealing.** The server may make one of its windows active only while one of its windows already is; a new window opened while the user is elsewhere shows as needing attention instead. A modal dialog blocks only its owner, never the rest of the device.
- **Full screen.** A server puts a window in full screen only with the `fullscreen` permission or after the user asked for it (the window's own full-screen action, or a key the client maps to it). The client shows how to leave full screen when it starts, and keeps a key to leave that the server never receives.
- **Moves the user started.** The `window` op is accepted only while a pointer button the user pressed in that window is still held, so a server cannot drag a window away by itself.
- **Limits.** The client's `max` bounds the windows open at once, `min` and `max` sizes bound each window, and a client may shrink a window that would not fit on its displays.

WINDOW input carries only what the user did to windows of this session: the client never reports its other windows, their titles or positions.

§11 Definitions and checks

The definitions live in the bundle under `ext/windows-1/`, next to the earlier extensions:

File	What it holds
`vtx-windows-1.cddl`	The capability, the WINDOW input frame, its actions and the `fullscreen` permission
`vtx-ops-windows-1.schema.json`	Every op of the core and the first nine extensions, plus the window node and the `window` op
`vtx-app-windows-1.schema.json`	Application lines with the same additions
`vtx-app-events-windows-1.schema.json`	Server lines to an application; the input event gains `window`

File	What it holds
`examples/valid/`	A window scene, the Evolution trace (main window, menu, frameless compose window, modal attach dialog, maximize, full screen, close), the full-screen permission trace and application lines
`examples/invalid/`	Broken scenes, schemas, application lines and traces, each with the check that must catch it

`tools/gen_ext_windows.py` writes the schemas, built on the Sensors 1 generator so every earlier part stays identical; `tools/make_windows.py` writes the examples. Both are deterministic.

The validator's Section 18 runs 73 checks, and every one passes:

Part	Checks	What they cover
Setup	6	The schemas are valid; the CDDL parses alone and with the first nine extensions; no rule is defined twice; earlier schemas and the core reject the additions
CDDL samples	8	4 valid samples accepted, 4 malformed ones rejected
Scene	33	13 valid lines, 13 broken by a scene rule (Sections 3, 5 and 6), 7 broken by the schema
Traces	16	2 valid traces replayed in full; 14 broken ones, from a missing capability to focus stealing and a window opened full screen unasked, each caught by the right check
Application interface	10	8 valid lines, 2 rejected

The whole bundle now runs 2,543 checks, all passing.

§12 Wire summary

Everything Windows 1 adds, for the registry and for implementers:

Kind	Addition
Capability	`surface.window`: `max` (default 16), `popups`, `frameless`, `icons`, `progress`
Node type	`window`: `title` and `size` required; `app`, `icon`, `role`, `owner`, `modal`, `at`, `offset`, `min`, `max`, `resizable`, `state`, `frame`, `progress`, `audience`
Op	`window`: `begin_move`, or `begin_resize` with an `edge`
Frame	0x1F WINDOW on channel 2 (input), client to server: `t`, `window`, `action`, `at`, `size`, `seen`
Window actions	`focus`, `blur`, `move`, `resize`, `minimize`, `maximize`, `restore`, `fullscreen`, `close`, `dismiss`
Permission	`fullscreen`
Application event	`{"kind":"input","event":"window","data":{…}}`, data as the WINDOW frame

No frame, op or field of the core or of an earlier extension changes meaning. A client or server that does not know Windows 1 never sees any of it, since the capability is never granted and the server falls back to the whole desktop (Section 8).