VTX Displays 1: multi-monitor sessions

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

§1 Status and scope

Displays 1 lets a session use every monitor of the device. A full remote desktop spans the displays the user picks, with one stream per display. Seamless windows open, move, maximize and go full screen on any display, each at that display's own scale. It is an extension of VTX 1, draft d01, in conformance profile 3, and the second step toward a full desktop session that replaces RDP.

It builds on two earlier extensions: Windows 1, which shows remote windows as the device's own, and Media 1, whose pixel surfaces and play streams carry the pictures. A server grants Displays 1 only together with `surface.window`.

Adds	What it is
`displays`	A capability: the client can report its displays; the server says how many a full desktop may use
LAYOUT	A control frame, 0x23 on channel 0, client to server: every display, where it sits and its scale, and which ones the user picked
`display`	A property of window nodes: the display a window opens on, or moves to
`layout`	An application event: the layout, passed on to the hosted desktop's compositor

This is what RDP calls "use all monitors" and "selected monitors", together with its dynamic monitor layout: plugging a display in or out, or changing its resolution or scale, takes effect at once.

Out of scope. HDR and colour profiles, which belong with Media 1's video formats. Mirrored displays, which the client reports as one. Hardware details such as model names and serial numbers, which never leave the device. The device's own virtual desktops (Spaces on macOS, workspaces elsewhere), which stay the client's business.

§2 Capabilities and limits

The client offers `displays` in HELLO; the server grants it in WELCOME, only together with `surface.window`, and may say how many displays a full desktop uses.

```
cap-displays = {
  ? "max": 1..16,              ; server: most displays a full desktop uses (default
16)
}
```

Who	Sends	Meaning
Client, in HELLO	`"displays": {}`	It can report its displays and place windows on them

Who	Sends	Meaning
Server, in WELCOME	`"displays": {"max": 4}`	It uses at most 4 displays for a full desktop: the first 4 the user picked, in the user's order

The layout itself is not in HELLO. A server learns about the device's displays only after it has granted the capability, in the LAYOUT frame the client sends right after READY (Section 3). A server that never grants it learns no more than v1's DISPLAY tells it.

A layout holds at most 16 displays. Seamless windows may go on any display in the layout; `max` limits only the full desktop, where every display costs the server a virtual output, an encoder and a stream.

§3 The display layout

The client describes all its displays as rectangles in one layout, in px, the device-independent pixels the scene already uses. The primary display's top-left corner is the origin; displays to its left or above it have negative positions. Each display has its own scale, the number of device pixels per px.

```
ctl-layout = {                                        ; 0x23 LAYOUT, C->S
  "displays": [1*16 display],
  ? "use": [1*16 display-id],   ; the displays the user picked for a full desktop, in
the user's order;
                                ; absent: the primary display alone
}
display-id = tstr .regexp "[a-z0-9-]{1,16}"
display = {
  "id": display-id,             ; chosen by the client for this connection, e.g. "d1"
  "at": [int, int],             ; top-left corner in the layout, px
  "size": [1..32767, 1..32767], ; px, as the user sees it: a portrait display is taller
than wide
  "scale": num,                 ; device pixels per px: 1, 1.25, 1.5, 2
  ? "hz": num,                  ; refresh rate
  ? "primary": true,
  ? "work": [int, int, uint, uint],   ; x, y, w, h: what the device's own taskbar, dock
or menu bar leaves free
}
```

A desk with a 27-inch 4K display in the middle at 150%, a 1080p display on its left and a 1080p display turned upright on its right sends:

```
{"displays": [
  {"id": "d1", "at": [0, 0], "size": [2560, 1440], "scale": 1.5, "hz": 60, "primary":
true, "work": [0, 0, 2560, 1392]},
  {"id": "d2", "at": [-1920, 360], "size": [1920, 1080], "scale": 1, "hz": 60},
  {"id": "d3", "at": [2560, -240], "size": [1080, 1920], "scale": 1, "hz": 60}],
 "use": ["d1", "d2", "d3"]}
```

The size is already as the user sees it, so rotation needs no field of its own: d3 is 1080 wide and 1920 tall.

Rules. A LAYOUT that breaks one of these is a protocol error.

1. Display ids are unique. The client picks them; they say nothing about the hardware.
2. Exactly one display is primary, and its `at` is [0, 0].
3. `scale` lies from 0.5 to 4, and `hz`, when present, is above 0.
4. Displays do not overlap. They may touch, and gaps between them are allowed.
5. A work area lies inside its display. Without one, the whole display is free.
6. `use` names displays of the layout, each once. Without it, a full desktop uses the primary display alone.

When it is sent. Right after READY, before any other frame on channel 0, when the server granted `displays`; again whenever the layout changes (Section 5); and again after every resume, since the user may come back on another device.

§4 At a glance

> **Diagram: full desktop on three displays · output, stream, window per display.** Drawn in the PDF version, `11-displays-1.pdf`.

The client's LAYOUT tells the compositor what the desk looks like; the compositor builds one output per picked display in the same places, and each output comes back as one stream into one full-screen window. The middle display runs at 150%, so its stream is 1.5 times as wide and as tall in device pixels as the display is in px.

§5 Layout changes

Every change sends the whole layout again, once it settles: the client waits until the user stops dragging displays around in the device's settings, then sends one LAYOUT. The server compares it with the last one and follows in its next UPDATE.

The user	Full desktop: the server	Seamless windows
Plugs a display in	Opens a window for it when the user adds it to `use` and `max` allows	Nothing moves; new windows may open there
Unplugs a display	Removes that display's window; the remote desktop moves its applications to the displays that remain	The device moves the windows that were on it, and the client reports each move
Changes resolution or scale	Resizes the display's window and surface, and re-encodes its stream at the new size times scale	Windows keep their size in px; their streams follow the new scale
Rearranges displays	Moves the remote outputs to match, so the pointer still crosses where it does on the desk	The positions of the windows change with their displays, and the client reports the moves
Changes `use`	Opens and removes windows to match	Nothing

Order. The client sends LAYOUT before the WINDOW frames for the moves it caused, so the server reads those positions against the new layout.

Nothing on a display that has gone. Once a LAYOUT has dropped a display, the server names it in no op, and the window it held for a full desktop is removed in the server's next UPDATE. Either one left undone is a protocol error.

Resuming elsewhere. A user who leaves a three-display desk and resumes on a laptop sends the laptop's layout with the resume. The full desktop shrinks to the one display; seamless windows that were on the others come back on the laptop's screen, as the device places them.

§6 The full desktop across displays

A full desktop is one Windows 1 window per display the user picked, each full screen on its display and holding one pixel surface fed by one stream. No new node type is needed: the window says which display, the surface says which stream.

```
{"op":"open","id":"w-d2","type":"window","title":"Desktop — left","display":"d2","at":
[-1920,360],"size":[1920,1080],"state":"fullscreen"}
{"op":"open","id":"s-d2","type":"surface","kind":"pixel","parent":"w-
d2","stream":"desk-d2","alt":"Remote desktop, left display","size":
[1920,1080],"fit":"contain","mouse":"move","keys":"raw"}
```

On the hosted desktop, the compositor creates one virtual output per picked display, with the same size, scale and position relative to the others. Remote applications therefore draw at the device's own scale, and a window dragged across the remote desktop crosses where the user's displays meet.

Rules for desktop windows.

1. A full-screen window that names a display is the whole of that display: its `at` and `size` are the display's own.
2. One such window per display.
3. At most `max` of them, on displays in `use`. The user picked those displays, which counts as asking for full screen there; a full-screen window on any other display needs the `fullscreen` permission, as in Windows 1.
4. The stream behind each surface has the display's size times its scale, in device pixels: 3840 × 2160 for the 4K display at 150%.

Leaving full screen. The client keeps a key to leave full screen on a display, as Windows 1 requires. That display's window then becomes an ordinary window with the device's own frame, which the user can move and resize, and the surface keeps its size and is scaled to fit, so the remote desktop does not rearrange itself. Going back to full screen restores it.

Which display has the keys. Clicking on a display makes its window the active one, and keys go there; the remote desktop decides which of its applications gets them. Displays not in `use` stay the device's own, so the user can keep local applications beside the remote desktop.

§7 Windows on several displays

With Displays 1, every `at` in Windows 1 is a position in the layout, and every `size` is in px. With one display at scale 1 that is exactly what Windows 1 means on its own, so nothing changes for a single screen.

The `display` property. A window node may name a display. On `open`, the window opens there; with an `at`, that point must lie inside the display, and without one, the client picks a place on it. On `patch`, the client moves the window to that display. Popups and tooltips never name one: they follow their owner.

```
{"op":"open","id":"w-compose","type":"window","title":"New
message","app":"org.gnome.Evolution","display":"d2","at":[-1500,560],"size":[800,600]}
```

Which display a window is on. The one that holds most of its area. The server works that out from `at` and `size` and the layout, so WINDOW needs no new field. A window the user drags to another display arrives as an ordinary `move`.

Scale. A window keeps its size in px when it moves to a display with another scale, but its picture does not: the compositor draws it again at the new scale, and its stream changes resolution in its next Media 1 CONFIG. Until then the client scales the old picture. A window that straddles two displays is drawn at the scale of the one holding most of it.

Maximize and full screen. Both happen on the window's own display. The client sends the new position with the new size, since both change:

Action	**`at`** and **`size`** in the WINDOW frame
`maximize`	The display's work area
`fullscreen`	The whole display
`restore`	Where the window was before

```
{"t":4200000,"window":"w-mail","action":"maximize","at":[0,0],"size":
[2560,1392],"seen":7}
```

A server that wants a window full screen on a given display patches `state` and `display` together, and needs the `fullscreen` permission or the user's request, as in Windows 1.

§8 Input across displays

Displays 1 adds no input frame. POINTER already carries a target and a position in px relative to that target, and every display's picture is a surface of its own, so the server always knows where on the remote desktop an event lands.

Input	Across displays
Mouse	Moving from one display to the next, the client sends `leave` on the first display's surface and `enter` on the next. The compositor maps each surface position through the layout, so the remote pointer crosses where the real one does

Input	Across displays
A drag	While a button is held, events keep the surface where the press began as their target, with positions past its edge. The compositor maps them through the layout, so a remote window dragged across displays is one continuous drag
Pointer lock	Motion is relative (`dx`, `dy`) and belongs to no display; the client keeps the local pointer on the display where the lock began
Touch	A touch belongs to the display it was made on, and targets that display's surface
Pen	The device maps the tablet to one display or to all; the client sends pen events against the surface under the pen
Keys	To the active window, whichever display it is on (Windows 1)

§9 With other extensions, and without displays

Extension	With displays
Media 1	One play stream per display for a full desktop, one per window for seamless applications, at the display's size times its scale. A display's `hz` caps the stream's frame rate, so a 144 Hz display can get more than 60 frames when the link allows. Screen capture stays as it was: the user picks a display or a window in the trusted zone, and the server never learns what the other displays show
Windows 1	`at` is a position in the layout, `display` places a window, and maximize and full screen carry the new position (Section 7)
Viewers 1	Each viewer sends its own LAYOUT. The remote outputs of a full desktop follow the owner's layout. Other viewers get the same streams, one window per remote display, placed on their own displays and scaled to fit; they go full screen only when they ask. Seamless windows are placed by each viewer's own device, as in Windows 1
Sensors 1	Each display's surface carries its own accessibility overlay, so a screen reader moves between remote displays as between windows
Transfer 1	Files and folders dropped on any display's surface arrive as on any pixel surface

Without the capability. A client without `displays` gets what Windows 1 gives on its own: one display, and a full desktop as one window sized from the core's DISPLAY. The user may still stretch that one window across several monitors, at one scale. A server that has not granted `surface.window` does not grant `displays` either, and the session is the single pixel surface of Surfaces 1.

§10 Application interface

The server passes every LAYOUT on to the application as a `layout` event with the same fields. On a hosted desktop, the compositor's VTX bridge receives it and shapes the remote desktop to match.

```
{"kind":"layout","displays":[{"id":"d1","at":[0,0],"size":
[2560,1440],"scale":1.5,"hz":60,"primary":true,"work":[0,0,2560,1392]},{"id":"d2","at":
[-1920,360],"size":[1920,1080],"scale":1,"hz":60},{"id":"d3","at":[2560,-240],"size":
[1080,1920],"scale":1,"hz":60}],"use":["d1","d2","d3"]}
```

A full desktop. The compositor creates one virtual output per display in `use`, up to `max`, with the display's position, size and scale. For each output it writes a desktop window, its pixel surface (Section 6) and a play stream at the output's device pixels:

```
{"kind":"stream","name":"desk-
d1","dir":"play","media":"video","codec":"av1","w":3840,"h":2160,"fps":60,"mode":"live"}
```

On each later `layout` event it adds, removes, moves and rescales outputs, and writes the window ops that follow. When the middle display goes from 150% to 125%, its output becomes 3072 × 1728 px; the window and surface are patched to that size, and the stream stays at 3840 × 2160 device pixels.

Seamless windows. The compositor keeps the remote window positions in the client's layout, so a window it places at a position lands on the matching display. When an application asks for a given screen, as a presenter view does for the second display, the compositor names that display in the window's `display`.

Mapping to the remote desktop. Wayland compositors give each output its own scale, so mixed scales render sharp everywhere. X11 has one scale for the whole screen; an X11 desktop renders at one scale and the client scales the pictures of the other displays.

§11 Security and privacy

A layout tells the server what the user's desk looks like, and full-screen windows cover real displays. The rules below keep both within what the user chose.

- **Only after the grant.** The layout is not in HELLO. A server sees it only once it has granted `displays`, so a server that has no use for it learns nothing new.
- **Nothing about the hardware.** Model names, serial numbers, EDID data, physical sizes, colour profiles and how a display is connected stay on the device. Display ids are made up by the client for each connection.
- **The user can share less.** A client may let the user report only some displays, down to the one the session runs on. Seamless windows then stay on those displays, and the server cannot tell the others exist.
- **Full screen only where the user said so.** Desktop windows are full screen only on the displays in `use`; anywhere else needs the `fullscreen` permission, as in Windows 1. The client keeps a key to leave full screen that the server never receives, and keeps the trusted zone reachable on every display a full desktop covers.
- **Nothing outside the session.** Pointer and touch events go only to the session's own surfaces, so the server never learns where the pointer is on displays it does not cover. Screen capture still goes through Media 1's `screen` permission.
- **Limits.** At most 16 displays per layout, and at most `max` of them for a full desktop. Every layout change makes the compositor reconfigure its outputs, so a server may apply layouts that arrive faster than one a second as one, taking the latest.

§12 Definitions and checks

The definitions live in the bundle under `ext/displays-1/`, next to Windows 1:

File	What it holds
`vtx-displays-1.cddl`	The capability and the LAYOUT frame with its displays

File	What it holds
`vtx-ops-displays-1.schema.json`	Every op of the core, the first nine extensions and Windows 1; window nodes gain `display`
`vtx-app-displays-1.schema.json`	Application lines with the same addition
`vtx-app-events-displays-1.schema.json`	Server lines to an application, plus the `layout` event
`examples/valid/`	Five layouts (a laptop, a laptop with a display, the three-display desk, displays with a gap, a 2 × 2 wall), a scene, the full-desktop and seamless traces, and the compositor's application lines
`examples/invalid/`	Broken layouts, scenes, application lines and traces, each with the check that must catch it

`tools/gen_ext_displays.py` writes the schemas, built on the Windows 1 generator; `tools/make_displays.py` writes the examples, built on Windows 1's. Both are deterministic.

The validator's Section 19 runs 82 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 and Windows 1; no rule is defined twice; earlier schemas reject `display`; no earlier CDDL has LAYOUT
CDDL and layouts	24	3 valid samples accepted and 6 malformed ones rejected; 5 valid layouts pass the CDDL, the rules of Section 3 and the event schema; 10 broken layouts pass the CDDL and fail a rule
Scene	18	13 valid lines, 3 rejected by the schema, 2 broken by the rule that popups and tooltips follow their owner
Traces	17	2 valid traces replayed in full; 15 broken ones, each caught by the right check, among them no grant, a bad layout, windows before the layout, an unknown display, a window outside its display, two desktop windows on one display, a wrong size, full screen on a display the user did not pick, more than `max`, a display gone, a stale scale
Application interface	17	15 valid lines, with every desktop stream at its display's size times scale; 2 rejected

The simulator replays every Displays 1 trace on top of the Windows 1 simulator, so all of Windows 1's rules hold too. Writing it closed a gap there: Windows 1 now also catches a window opened full screen without the permission, in a fourteenth broken trace. The whole bundle runs 2,543 checks, all passing.

§13 Wire summary

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

Kind	Addition
Capability	`displays`: the client sends `{}`; the server answers with `max` (1 to 16, default 16)
Frame	0x23 LAYOUT on channel 0, client to server: `displays` (1 to 16, each `id`, `at`, `size`, `scale`, `hz`, `primary`, `work`) and `use`

Kind	Addition
Window property	`display` on window nodes (Windows 1), never on popups or tooltips
Application event	`{"kind":"layout","displays":[…],"use":[…]}`, the fields of the last LAYOUT
Meaning of earlier fields	With Displays 1, Windows 1's `at` is a position in the layout; `maximize`, `fullscreen` and `restore` carry `at` as well as `size`

No frame of the core, the first nine extensions or Windows 1 changes meaning, and no permission is added: full screen on the displays the user picked counts as the user's request, and anywhere else uses Windows 1's `fullscreen`.