VTX Wayland Host 1: a Wayland compositor as a VTX server

Nov 15, 2026 · VTX 1 · host binding · MIT licence · vtx.foundation/spec/v1/wayland-host-1/

§1 Status and scope

Wayland Host 1 is a host binding of VTX 1, draft d01. It is not an extension: it adds no capability, no frame and no channel. It pins down how a Wayland compositor on a session host presents its applications through the VTX definitions that already exist, so that every host of the hosted Linux desktop behaves the same way and a client can rely on it.

The binding covers two ways of showing a session:

- **Full desktop.** The compositor draws a whole desktop on one virtual output per display the user picked, and each output becomes one full-screen window with one video stream, as Displays 1 describes. Applications, panels and notifications live inside it, as on a local machine.
- **Seamless.** Each application window becomes one of the device's own windows, as Windows 1 describes: the device's frame, its taskbar and window switching, menus and tooltips placed against their owner. The compositor draws no desktop of its own.

It works on three layers, each in its own sections: outputs and windows (Sections 3 and 4), pixels (Section 5) and input (Section 6). Section 7 covers the clipboard and the other channels, and Section 8 covers X11 programs, locking and what the compositor refuses to applications. The parts that reach the wire are also written down as an executable reference mapping, and the binding's example logs must match it exactly (Section 9).

Out of scope. Which compositor to build on (wlroots, Smithay, Mutter or KWin) is an implementation choice; the binding only says what each of them must do. Running a Windows host is the RDP bridge's subject, the next item on the roadmap.

§2 At a glance

> **Diagram: the seamless mapping · seven kinds of Wayland object and what each becomes.**
> Drawn in the PDF version, `01-wayland-host-1.pdf`.

The client's displays become the compositor's outputs; application windows, menus and tooltips become the device's own windows, each fed by its own stream; size changes go through the application before the scene changes; input arrives from the device; the selection goes through the clipboard channel. What only makes sense inside a desktop, or should never leave the host, is not sent. In a full desktop, the first row is all there is on the wire: each output is one window and one stream, and everything else stays inside it.

§3 Outputs and displays

The compositor's outputs are the client's displays. When LAYOUT arrives, the compositor creates one virtual output for each display, or changes or removes the ones it has; it has no physical outputs of its own in a VTX session.

Output property	From the display in LAYOUT
Logical position and size (zxdg_output_v1)	`at` and `size`: VTX px are Wayland's logical pixels, and both put the primary display at 0, 0
Preferred scale (wp_fractional_scale_v1)	`scale`, sent as `scale` × 120
Integer scale (wl_output.scale, preferred_buffer_scale)	`scale` rounded up, for applications without fractional scaling
Mode	`size` × `scale` device pixels, at `hz`
Work area, for maximizing	`work`

Which outputs. In a full desktop the compositor creates outputs only for the displays in `use`, the ones the user picked, and each becomes one full-screen window and one stream (Displays 1). In seamless mode it creates an output for every display, so applications know at what scale to draw wherever their window is. A seamless window's display is where the compositor placed it last; when the user moves a window with the device's frame, the compositor follows the position WINDOW reports.

Changes. Every LAYOUT carries the whole layout, so the compositor compares and acts: a new display gets a new output, a changed one changes its output (applications get a new scale and redraw), and a display that is gone loses its output. In seamless mode, windows on a display that is gone move to the primary display, and the compositor tells the client with a patch of their `display`; the client places them. In a full desktop, the desktop window of a display that is gone is removed, and the compositor moves its applications to the outputs that remain, as a desktop does when a monitor is unplugged.

Resolution. Each output, or each seamless window, is drawn at its display's scale, so the stream's coded size is the size in px times the scale: a 2560 × 1440 px display at 150% gets a 3840 × 2160 stream. When that exceeds the client's `video_px` limit, the compositor draws at a lower scale and the client scales the picture up.

§4 Windows

In seamless mode every xdg-shell toplevel and popup becomes a window node with one pixel surface in it, fed by a stream of its own. The compositor opens it when the application first commits a buffer, and removes it when the application unmaps or destroys it.

xdg-shell	Windows 1
set_title	`title`, and the surface's `alt`
set_app_id	`app`, which groups windows on the device's taskbar
set_parent	`role` dialog and `owner` the parent's window
xdg_dialog_v1.set_modal	`modal`

xdg-shell	Windows 1
set_min_size, set_max_size	`min`, `max`
The window geometry of the first commit	`size`, and `display`: the parent's display, else the active window's, else the primary one. `at` is left out, so the client places the window as it places its own
xdg_popup with a grab	`role` popup, `owner` the parent's window, `offset` where the positioner put it
xdg_popup without a grab	`role` tooltip, the same way
Server-side decorations (zxdg_decoration_manager_v1)	`frame` native, the default: the device's own frame
An application that draws its own decorations	`frame` none, so the client asks the server to move it

Popups name no display: they follow their owner. A popup of a popup, a submenu, is owned by the popup it came from.

Sizes go through the application. A Wayland application decides its own size: the compositor configures it, and the application commits a buffer of the size it settled on. So every size change the user makes with the device's frame (resize, maximize, restore, full screen) reaches the application as a configure, and the compositor patches the window's `size`, and `state` and `at` where they changed, only after the application has committed the new size. The scene then never shows a size the application has not drawn. A size the application changes on its own reaches the scene the same way.

What the application asks for. set_maximized and unset_maximized configure the application to the display's work area or back, and the patch follows its commit; set_minimized becomes `state` minimized at once. set_fullscreen needs the `fullscreen` permission: the compositor asks the user on the trusted zone channel, and only after `allow` configures the application to the whole display. A grant for the session or always covers later requests; a grant for once covers this one.

Moves and resizes. An application that draws its own title bar asks for an interactive move or resize with xdg_toplevel.move or resize. The compositor honours it only while a button is held on that window, as every compositor checks the input serial, and sends the `window` op with `begin_move` or `begin_resize`; the device's window manager does the rest. Without a held button, the request is ignored.

Focus. The device decides which window is active, and WINDOW `focus` and `blur` become the toplevel's activated state and keyboard focus. An application may raise a window with xdg_activation_v1 only with a token from the user's own input in the session, and only while one of the session's windows is active; the compositor then sends `focus`. Any other activation becomes `attention`, which the device shows as it shows a program that wants the user. No application can take focus away from what the user is doing on the device.

Closing. The close button sends WINDOW `close`, which becomes xdg_toplevel.close; the window stays until the application unmaps it, so an application can still ask to save first. WINDOW `dismiss` on a popup becomes popup_done.

§5 Pixels

What each stream holds. In a full desktop, each output is composed as a local compositor composes a monitor and becomes one video stream (`desk-` and the display id). In seamless mode, each window is composed on its own: the toplevel's surface and its subsurfaces, cut to its window geometry, become one stream (`win-` and the window), and each popup gets its own. Shadows an application draws outside its geometry are left out, because the device draws its own frame and shadow.

Encoding. The compositor encodes from the GPU buffers it composes into (dmabufs through VA-API, NVENC or a software encoder), in the first codec of the client's `video` decode list it can produce, at the output's or the window's size times the display's scale. Damage tells the encoder what changed: an unchanged window sends nothing, a blinking caret sends a few blocks. FEEDBACK sets the bitrate and asks for key frames; a window that just opened, or whose size changed, starts with one.

Pacing. The compositor releases wl_surface.frame callbacks at the pace it can send frames, never faster than the display's `hz` and never faster than FEEDBACK lets it send, so applications draw as often as the device can show and no more. When VIEW says a window is not visible, its applications get no frame callbacks and the toplevel gets the suspended state, so hidden applications stop drawing. wp_presentation reports when a frame was sent.

Cursors. The client draws the pointer, so moving it never waits for the network. A shape an application names through wp_cursor_shape_device_v1 becomes the surface's `cursor`, the device's own pointer shape: `default`, `text`, `pointer`, `grab`, `grabbing`, `crosshair`, `move`, `not-allowed`, `wait`, `progress`, `help` and the four resize shapes map one to one, and the other names to the closest of these. A cursor image set with wl_pointer.set_cursor becomes an image resource with its hot spot. The cursor is never drawn into the stream.

Colour. Streams use BT.709, Media 1's default. Applications that use the colour management protocol for HDR or wide gamut are converted to BT.709; HDR streams are left to a later draft.

§6 Input

Input comes from the device through v1's input channel and reaches applications through the compositor's seat, as if a keyboard, mouse, touchscreen and pen were plugged into the host.

VTX input	Wayland
The keyboard's `layout` (HELLO or DEVICES)	wl_keyboard.keymap: an XKB keymap for that layout, so `code`, the physical key, gives the same character in the session as on the device
KEY `code` and `mods`	wl_keyboard.key and modifiers, from the physical key code
KEY `repeat`	A release and a new press: the compositor sends repeat_info with rate 0, so applications do not repeat on their own and a late release over the network cannot make a key run on
POINTER on a pixel surface	wl_pointer.motion and button, in surface coordinates: VTX px are Wayland's logical pixels
WHEEL `px` and `line`	wl_pointer.axis for smooth scrolling, axis_value120 in steps of 120 a line
POINTER `touch`	wl_touch, one touch point per contact `id`
POINTER `pen`	zwp_tablet_tool_v2, with pressure, tilt, twist and hover from `contact`

VTX input	Wayland
COMPOSE	zwp_text_input_v3: `update` as the preedit string, `commit` as the commit string

Pointer lock. An application that locks or confines the pointer (zwp_pointer_constraints_v1, with zwp_relative_pointer_v1 for the motion) sets the surface's `pointer` to lock, behind v1's `pointer.lock` permission; relative motion arrives as `dx` and `dy`. Games and 3D programs need it, and the user can always end it on the device.

Shortcuts. System shortcuts such as Alt+Tab stay with the device unless the user granted `keyboard.lock`. An application that asks to receive them (zwp_keyboard_shortcuts_inhibit_manager_v1, as virtual machine viewers do) gets them only on a surface with `keys` all, after that grant.

Text input. The device's own input method composes text, so East Asian input and dictation work as the user set them up on the device. VTX does not yet tell the client where the caret is in a pixel surface, so the input method's candidate window appears where the device puts it; a caret position for pixel surfaces is a candidate for a later draft.

§7 Clipboard, drag and drop, and the other channels

Clipboard. When an application in the session sets the selection (wl_data_device.set_selection), the compositor reads the types it can carry, text and images first, and writes them to the device's clipboard with WRITE on the clipboard channel, under v1's `clipboard.write` rules. When an application pastes, the compositor offers the device's clipboard as the session's selection, reading it with READ under `clipboard.read`; files on the clipboard go through Transfer 1's FETCH. The primary selection, middle-click paste, stays inside the session.

Drag and drop. Inside the session, drag and drop is the compositor's own business. Between a seamless window and the device's desktop, files travel as Transfer 1 uploads from a drop, so a file dragged from the device's file manager into a session window arrives as a file the application can open.

The other channels. Everything else a desktop needs comes from the other extensions, and the compositor only has to connect them:

Need in the session	Through
Sound from applications, and their microphones and cameras	Media 1 streams, bound to the device's devices with Media Devices 1
Opening and saving files on the device	Drives 1, with the file chooser portal pointing at the shared folders
Printing	Printers 1
Smart cards and security keys for sign-in	Smart Cards 1
Other USB devices	USB 1
Notifications	Devices 1's notices, or a full desktop's own notification panel
Opening a link on the device	v1's `open_url`

Screen sharing in the session. An application that shares the screen, in a video call for example, goes through the screen cast portal, which asks the user and offers only the session's own windows and outputs. It never sees the device's screen; sharing the device's screen is Media 1's `screen` capture, which the device asks for itself.

§8 X11 programs, locking, and what the compositor refuses

X11 programs. Older programs run under Xwayland without a root window, and the compositor's X11 window manager maps their properties onto the same Windows 1 properties, so they look and behave like every other seamless window:

X11	Windows 1
_NET_WM_NAME, or WM_NAME	`title`
WM_CLASS	`app`
WM_TRANSIENT_FOR	`owner`, with `role` dialog
_NET_WM_WINDOW_TYPE dialog, utility	`role` dialog, utility
Override-redirect windows of type menu, dropdown menu, popup menu, combo	`role` popup, owned by the window they belong to
Override-redirect windows of type tooltip	`role` tooltip
_MOTIF_WM_HINTS without decorations	`frame` none
_NET_WM_STATE_FULLSCREEN	A full screen request, behind the `fullscreen` permission
_NET_ACTIVE_WINDOW from the program	`attention`: X11 has no activation token

Locking. When the session locks (ext_session_lock_v1), nothing of the applications may show. In a full desktop the lock screen fills the outputs as usual. In seamless mode the compositor hides every window (`hidden`) and opens one window holding the lock surface; unlocking removes it and shows the windows again.

What the compositor refuses. On a local desktop, every application belongs to the person at the keyboard. In a hosted session, a compromised application must not reach what the user does elsewhere, so the compositor refuses these protocols to applications in the session:

Protocol	Why
zwlr_screencopy_manager_v1, ext_image_copy_capture_manager_v1	Reading the screen behind the user's back; screen capture goes through the portal, which asks
zwp_virtual_keyboard_manager_v1, zwlr_virtual_pointer_manager_v1	Typing and clicking as if they were the user; input comes from the device
zwlr_foreign_toplevel_manager_v1	Listing and controlling other applications' windows
zwlr_data_control_manager_v1, ext_data_control_manager_v1	Reading and setting the clipboard in the background
zwp_input_method_manager_v2	Composing text in place of the device's own input method

zwlr_layer_shell_v1 is not refused but has nothing to show in seamless mode: panels, docks and notifications from the session stay out, because the device has its own.

§9 Definitions and checks

As a binding, Wayland Host 1 lives in the bundle under `hosts/wayland-1/`, beside the extensions rather than among them, and is not part of the registry:

File	What it holds
`wayland-host-1.json`	The mapping: 48 rows, each a Wayland protocol or request, its level (required, optional, desktop, host or refused), the VTX definitions it maps to and why
`examples/valid/log-seamless.jsonl`	The seamless log below: 23 things happening in the compositor and 38 frames
`examples/valid/log-desktop.jsonl`	A full desktop on two displays: a scale change, and the second display unplugged and plugged back
`examples/invalid/logs.jsonl`	The logs broken 13 ways, one at a time

`tools/make_wayland.py` writes them, with every server frame written out by hand. `tools/wayland_host.py` is the reference mapping: given what happens in the compositor and the client's frames, it produces the frames a host sends. A log is a header line, then the compositor's events (`{"wl": …}`) and the frames on the wire, in order.

The seamless log. LibreOffice Writer opens on the laptop, its cursor and title follow, a menu and a tooltip open against it and close, and a modal Save As comes and goes. Writer asks for full screen; the compositor asks Marcel first and patches only after Writer has drawn the full display. Marcel drags Writer to the external display and maximizes it there, and the patch waits for Writer's commit. Files, which draws its own headerbar, opens on that display, takes focus with an activation token, and is dragged by its headerbar; a second move request after the button is released is ignored. Calendar's reminder asks for focus without a token and gets attention instead. A notification panel stays out, Files closes, and the display is unplugged: the windows move to the laptop, and Writer is maximized there again.

The validator's Section 26 runs 67 checks, and every one passes:

Part	Checks
The mapping's shape: every row with its protocol, level, targets and reason	1
Each row's targets exist: ops and window and surface properties in the schemas, frames, fields, window actions and permissions in the CDDL	48
The reference mapping handles exactly the compositor events the rows name	1
Each valid log is exactly what the reference mapping sends, and keeps every rule of Windows 1 and Displays 1	4
Each broken log fails for its labelled reason	13

The broken logs: a title change not forwarded; a menu opened as an ordinary window; full screen without asking; an activation without a token that took focus; a move with no button held; a maximize patched before the application drew it; a new window away from the active window's display; a notification panel shown as a seamless window; a modal dialog without its owner; windows left on an unplugged display; a compositor event the binding does not map; a desktop window left for an unplugged display; and an application window sent as a seamless window in a full desktop.

The binding adds nothing to the wire, so no other count changes. The whole bundle runs 2,543 checks, all passing.