VTX Drives 1: the device's folders in a session

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

§1 Status and scope

Drives 1 lets a session work on folders of the user's own device, live: a program on the host opens, edits and saves a file that stays on the device, as RDP's drive redirection does with `\\tsclient`. Where Transfer 1 copies files across, Drives 1 never makes a copy the user has to bring back. It is an extension of VTX 1, draft d01, in conformance profile 3, and the first of the redirections under the `redirect.` prefix.

Adds	What it is
`redirect.drives`	A capability: the client can share folders; the server can mount them
`drives`	A permission: the server asks the user to share a folder
`drive` channel	One channel per shared folder, opened by the client. The server sends file requests; the client answers them from the folder
`drive`	An application event: a shared folder was mounted or removed

The device is the file server and the host the file client. The host shows each shared folder to its programs as an ordinary folder, through the file system bridge of its own platform (Section 8).

Out of scope. Byte-range locks: programs that need them keep using lock files, as they do on network shares. Access control lists, extended attributes and alternate data streams. Hard links, special files and sharing whole devices such as USB sticks below the file level. Printers, microphones and smart card readers, which are the redirections that follow.

§2 Capabilities, the permission and sharing a folder

Only the user shares a folder, and only the client opens a drive channel. The server may ask; it can never reach a folder on its own.

```
cap-drives = {
  ? "max": 1..16,            ; client: most folders shared at once (default 4);
server: most it mounts
  ? "watch": bool,           ; client: it reports changes in watched folders
}
```

Two ways to share.

Who starts it	What happens
The user	Picks a folder in the client's own menu, in the trusted zone, and chooses read-only or read and write
The server	Sends v1's PERMISSION with `drives` and a `detail` saying why. The client lets the user pick folders, or decline

The permission's scope works as in v1: `once` shares for this connection, `session` until the session ends, and `always` makes the client share the same folders with this server each time it connects, until the user revokes it.

A share is a channel. The client opens one channel of type `drive` per folder; its OPEN params name the share:

```
OPEN {"channel": 21, "type": "drive",
      "params": {"name": "Documents", "access": "write", "case": "insensitive", "free":
182536110080}}
```

OPEN_OK means the server has mounted it; OPEN_REFUSED means it will not, for instance because it already mounts its `max`. To stop sharing, the client sends v1's CLOSE on the channel: the server removes the folder from the session, and programs that still have files open there get the error `stale`.

Rules. At most `max` shares per connection, the lower of the two sides' values. Share names are unique per connection, so a second "Documents" is renamed by the client, for instance to "Documents (2)".

§3 Paths, names and attributes

Every path is relative to the shared folder. The empty path is the folder itself; `Invoices/invoice-0412.odt` is a file two levels down. The server never sees where the folder sits on the device.

```
drv-path = tstr .regexp "([^/]{1,255}(/[^/]{1,255})*)?"   ; no NUL, names at most 255
bytes (rules)
drv-attrs = (
  "type": "file" / "dir",
  "size": uint,               ; bytes; 0 for a folder
  "mtime": millis,            ; milliseconds since 1970-01-01 UTC
  ? "readonly": true,
  ? "hidden": true,
)
```

Names. UTF-8 in Unicode normalization form C, at most 255 bytes, never `.` or `..`, and a whole path at most 4,096 bytes. A client on a device that stores another form, such as macOS with its decomposed names, converts at its edge. A name the device cannot store, such as `a:b` or `CON` on Windows, gets the error `name`.

Case. The share's `case` says how the device compares names. On an `insensitive` share, `Invoice.odt` and `invoice.odt` are one file, and the host's mount must behave the same way, or a program could create two files the device holds as one.

What is visible. Files and folders only. A symbolic link that stays inside the shared folder is followed; one that points outside it looks like a missing file. Devices, sockets and pipes are never listed.

Attributes. Type, size, the time of the last change, and two flags: `readonly` for a file the device will not let the user change, and `hidden` for one the device hides, such as a dot file or a Windows hidden file. Owners, permission bits and access control lists stay on the device: on the host, every file belongs to the session's user.

§4 At a glance

> **Diagram: a file call from a program on the host to a folder on the device.** Drawn in the PDF version, `13-drives-1.pdf`.

The program uses an ordinary folder on the host; the server turns its file calls into drive requests, and the client answers them from the one folder the user shared. The share check on the device is the only way in, so a server that sends a wrong path or a write to a read-only share is stopped there, whatever it sends.

§5 File operations

The server sends requests, each with a number that grows; the client answers every one exactly once, with RESULT or ERROR, in any order. Up to 64 requests may wait at once, so a program reading a large file keeps the link full.

Request	Asks for	RESULT carries
0x10 STAT	A path's attributes	The attributes; for `""`, also `free`
0x11 LIST	A folder's entries, a page of at most 256 after `after`	`entries`, and `more` when another page follows
0x12 CREATE	Open a file, or create it (`create`: `never`, `new`, `always`; `truncate`)	A `handle` and the file's attributes
0x13 READ	Up to 65,536 bytes at an offset	`data`, shorter only at the end of the file
0x14 WRITE	Bytes at an offset	`written`
0x15 RELEASE	Close a handle	Nothing more
0x16 MKDIR	A new folder	Nothing more
0x17 REMOVE	Delete a file or an empty folder	Nothing more
0x18 RENAME	Move a file or folder, replacing the target only with `replace`	Nothing more
0x19 TRUNCATE	Set an open file's size	Nothing more
0x1A WATCH, 0x1B UNWATCH	Report changes in a folder (Section 6)	Nothing more
0x1C SETTIME	Set a path's time of last change	Nothing more

Handles. CREATE opens a file for `read`, `write` or `readwrite` and returns a handle; READ, WRITE and TRUNCATE use it within that mode; RELEASE ends it. The client keeps the file open on the device in between, so the device's own programs see it in use, as they would a file open on a network share.

Saving safely. Programs that save by writing a new file and renaming it over the old one work as they do locally: CREATE the new file with `always` and `truncate`, WRITE it, RELEASE it, then RENAME it with `replace`. If the link drops halfway, the old file is untouched.

Errors.

code	Meaning
`notfound`, `exists`	No such path; the path already exists
`notdir`, `isdir`, `notempty`	Wrong kind of entry for the request; a folder that still has entries
`denied`, `readonly`	The device refuses; the share or the file is read-only
`name`	The device cannot store that name
`space`, `toolarge`	The device's volume is full; the file would exceed what it can hold
`busy`	Another program on the device holds the file
`stale`	The handle or share has ended
`io`	The device's storage failed

§6 Watching, caching and dropped connections

Watching. WATCH asks the client to report changes in one folder, not its subfolders. When a file appears, disappears, is renamed or changes there, the client sends CHANGED with the folder and, when it knows them, the names; without names, the server lists the folder again. A file manager on the host then shows a file the user saved on the device a moment later. A client that did not offer `watch` sends no CHANGED, and the server polls an open folder at most once a second instead.

```
CHANGED {"path": "Invoices", "names": ["scan-0413.pdf"]}
```

Caching. The server may keep attributes and listings for at most one second, or until CHANGED for a watched folder. It may keep data it read while the handle stays open, and drops it when the file's size or time changes. It caches no writes beyond those already sent: when RELEASE is answered, the client has handed every write to the device's file system, and the file is on the device.

When the connection drops. The share channels end with the connection, and every handle with them. While the session waits to be resumed, the host's mount stays in place and programs' requests wait, up to the host's own limit. After the resume, the client opens the share channels again for every folder shared with scope `session` or `always`. The server may then reopen a handle by its path and mode if STAT shows the file unchanged; otherwise the program gets `stale`, as it would if a network share went away.

Several clients. With Viewers 1, only the owner's connections share folders, so the session's programs, which run as the owner, never reach another viewer's device. A folder shared from a second device of the same owner is a separate share, named as that device names it.

§7 With other extensions, and without the capability

Extension	With shared folders
Transfer 1	Copies stay copies: a file dragged onto a remote window, or pasted into one, still arrives as an upload, even from a shared folder. Drives 1 is for working on a file where it lies
Brokering 1	A file opened with a published item still travels as an upload in draft d01, so saving needs a download. A later draft can let LAUNCH name a file on a shared folder, so the program saves straight back to the device

Extension	With shared folders
Exec 1	Commands run in the session see the shared folders like any other folder
Viewers 1	Only the owner's connections share folders (Section 6)
Windows 1, Displays 1	Nothing changes: a file dialog in a remote window shows the shared folders where the host puts them

Without the capability. A client or server without `redirect.drives` falls back to Transfer 1: the user uploads the files a program needs and downloads its results, through the trusted zone each time.

§8 Application interface

The server, not the application, mounts a shared folder. It turns the host's file system calls into drive requests through the bridge its platform offers: FUSE on Linux, WinFsp or Projected File System on Windows, and a File Provider extension on macOS. Programs then use the folder with their ordinary file calls, and no file data ever crosses the application interface.

Host	Where a share appears
Linux	`/run/user/<uid>/vtx/drives/<name>`, in the session user's runtime folder
Windows	A network location, `\\vtx\<name>`, and a drive letter when the administrator allows
macOS	A location in Finder's sidebar under the server's name

The server mounts without set-user-ID bits and without device files, and on an `insensitive` share it compares names without case. It mounts without execute permission unless the host's administrator allows running programs from the user's device.

The application, usually the session manager or the desktop shell, learns about shares through `drive` events, so it can show them in the file manager's sidebar and in file dialogs:

```
{"kind":"drive","action":"added","name":"Documents","path":"/run/user/1000/vtx/drives/
Documents","access":"write"}
{"kind":"drive","action":"added","name":"Pictures","path":"/run/user/1000/vtx/drives/
Pictures","access":"read"}
{"kind":"drive","action":"removed","name":"Pictures"}
```

§9 Security

A shared folder opens part of the user's device to programs on a machine the user does not control. The client is the one that enforces every limit, because only the client can be trusted to.

- **Only what the user picked.** The client answers from the shared folder alone. It resolves every path below that folder, refuses `.` and `..`, and treats a link that leads outside it as missing. The server never learns where the folder sits on the device.
- **Read-only means read-only.** On a `read` share the client refuses every change with `readonly`, whatever the server sends; a server that sends one breaks the protocol.

- **Always in sight.** The trusted zone lists every shared folder for as long as it is shared, with its access, and stops sharing with one click. Sharing `always` is remembered per server identity, and the user can revoke it in the client.
- **Asked, never taken.** The server can ask with the `drives` permission; it cannot open a drive channel itself. Only the owner's connections share folders.
- **Nothing runs by default.** The host mounts shared folders without set-user-ID bits, device files or execute permission unless its administrator allows running programs from the user's device (Section 8).
- **Bounded work.** At most `max` shares (16 at the most), 64 requests waiting per share, 65,536 bytes per READ, 256 entries per LIST page. v1's channel credits pace the data both ways.

§10 Definitions and checks

The definitions live in the bundle under `ext/drives-1/`:

File	What it holds
`vtx-drives-1.cddl`	The capability, the `drives` permission, the share params, and the requests, answers and changes on a drive channel
`vtx-app-events-drives-1.schema.json`	Server lines to an application, plus the `drive` event. Drives 1 adds no scene op and no application line
`examples/valid/`	The Documents trace (open an invoice, save it through a temporary file and a rename, read it back, see a new scan, share Pictures read-only when GIMP asks, stop sharing) and the drive events
`examples/invalid/`	Broken drive events and traces, each with the check that must catch it

`tools/gen_ext_drives.py` writes the schema, built on the Brokering 1 generator; `tools/make_drives.py` writes the examples. File contents in the traces are placeholders of known bytes, so the checker can compare what a program reads back with what it wrote. Both tools are deterministic.

The validator's Section 21 runs 47 checks, and every one passes:

Part	Checks	What they cover
Setup	6	The schema is valid; the CDDL parses alone and with every earlier extension; no rule is defined twice; earlier schemas reject `drive`; the core alone rejects the `drives` permission
CDDL samples	17	8 valid samples accepted, 9 malformed ones rejected
Traces	19	The valid trace replayed in full, with every read-back compared byte for byte; 18 broken ones, each caught by the right check, among them a path that climbs out of the folder, a change to a read-only share, a read on a released handle, an answer of the wrong shape, a read-back that differs from what was written, and a request after the user stopped sharing
Application interface	5	3 valid drive events, 2 rejected

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

§11 Wire summary

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

Kind	Addition
Capability	`redirect.drives`: `max` (1 to 16; default 4) and, from the client, `watch`
Permission	`drives`: the server asks the user to share a folder
Channel type	`drive`, opened by the client, one per shared folder; OPEN params `name`, `access`, `case`, `free`, `total`
Requests, server to client	0x10 STAT, 0x11 LIST, 0x12 CREATE, 0x13 READ, 0x14 WRITE, 0x15 RELEASE, 0x16 MKDIR, 0x17 REMOVE, 0x18 RENAME, 0x19 TRUNCATE, 0x1A WATCH, 0x1B UNWATCH, 0x1C SETTIME
Answers, client to server	0x20 RESULT, 0x21 ERROR, 0x22 CHANGED
Error codes	`notfound`, `exists`, `notdir`, `isdir`, `notempty`, `denied`, `readonly`, `name`, `space`, `busy`, `stale`, `io`, `toolarge`
Application event	`{"kind":"drive","action":"added" or "removed","name":…,"path":…,"access":…}`

No frame of the core or of an earlier extension changes meaning. Stopping a share is v1's CLOSE on its channel.