VTX Composites 1: custom components

Nov 15, 2026 · VTX 1 · extension · draft d02 · MIT licence · vtx.foundation/spec/v1/composite-1/

§1 Status and scope

Composites 1 lets an application define its own components out of core components, with props, local variables and a small state machine, and use them as often as it likes. The client runs them locally, so expanding a card, flipping a toggle or timing out a confirmation needs no round trip. There is no code: a composite is data the client interprets. This is draft d02 of an extension of VTX 1, in conformance profile 2, built on the core at draft d03; d02 adds nesting (Section 6).

- **Always available.** A client without the capability still gets composites: the server expands them into core components and runs their state machines itself (Section 8).
- **Not in this draft.** Composites from other catalogs as parts, and composites that draw on surfaces. WASM components are WASM 1.

§2 Capability, composite catalogs and instances

Composites ride on v1's catalog scoping: a composite catalog is one more catalog a region can declare under a prefix.

- **Capability.** `composites: true` says the client runs composites itself.
- **Resource kind `catalog/vtx`.** A composite catalog (Section 3) in deterministic CBOR (RFC 8949, Section 4.2.1), offered and cached like any resource.
- **Declaring.** A region's `catalogs` maps a prefix to the catalog's resource hash, as it maps prefixes to catalog URIs in v1 Section 9.
- **Instances.** An instance is a node of type `prefix:name` whose props are the composite's props. As for every prefixed type, it carries `fallback`. `patch` changes its props, and the new `signal` op sends an event to its state machine (Section 5).

```
{"op":"open","id":"ops","type":"region","title":"Fleet","catalogs":
{"fl":"sha256:e901cf1e…"}}
{"op":"open","id":"web03","type":"fl:status-card","parent":"ops","fallback":"web-03:
critical","title":"web-03","status":"critical","detail":"502 errors from upstream"}
{"op":"signal","id":"web03","event":"restarting"}
```

§3 Defining a composite type

A composite catalog is `{ name, types }`, with up to 256 types named like `status-card`. Each type has:

Field	Meaning
`props`	Up to 64 props, each `{ type, enum, default, required, description }` with type `string`, `number` or `boolean`

Field	Meaning
`vars`	Initial local variables, a JSON object
`template`	1–256 parts: v1 `open` ops for core catalog types, as an application would write them, or nested composites from the same catalog (Section 6). A part without `parent` hangs off the instance; a `parent` names an earlier part. Part ids are unique within the type
`machine`	The state machine (Section 5)
`value`	A `/vars` pointer; when present, the composite is a form field (Section 7)
`fallback`	Text with `{prop}` placeholders, used where the composite cannot be shown

Parts are ordinary components, so they keep their semantics, accessibility and theming. Regions and grids cannot be parts.

§4 The local model and bindings

Each instance has its own data model, `{ props, vars, state }`, and every binding inside its template resolves there, not in the region's model.

- `/props/<name>` holds the instance's props: what was set with `open` or `patch`, else the default.
- `/vars/…` holds the local variables, which inputs inside the template write to.
- `/state` holds the name of the current state.

A binding outside these three is invalid, which keeps a composite self-contained: it sees only what its instance is given. To show region data, the instance binds its own props to the region model, `"title": {"bind": "/agents/0/name"}`, and the template binds to `/props/title`.

§5 The state machine

A machine has an `initial` state and up to 32 named `states`. In each state:

- **`set`** gives props for parts while the machine is in that state, such as `{ "detail": { "hidden": false } }`. Leaving the state restores the template's props.
- **`on`** maps an event to a target state, either as a name or as `{ to, emit }`. With `emit`, taking the transition also sends an INTENT with that name to the application.
- **`after`** `{ ms, to }` moves to a state after 100 ms to 24 hours in this state; the timer restarts each time the state is entered.

Events come from two places: an `action` part whose `intent` is an event some state handles fires it locally, and the server sends one with `signal`. An event the current state does not handle is ignored. A `signal` naming an event no state handles is `invalid_op`.

```
"machine": {"initial": "collapsed", "states": {
  "collapsed": {"on": {"toggle": "expanded"}},
  "expanded": {"set": {"detail": {"hidden": false}, "restart": {"hidden": false},
"more": {"label": "Hide details"}},
               "on": {"toggle": "collapsed", "restarting": "busy"}, "after": {"ms":
60000, "to": "collapsed"}},
  "busy": {"set": {"badge": {"state": "pending", "text": "restarting"}, "more":
{"disabled": true}},
           "on": {"done": "collapsed"}, "after": {"ms": 30000, "to": "collapsed"}}}}
```

Resume. The client keeps each instance's state and variables in its replica, so a resume that replays updates keeps them. A reset snapshot starts every instance again in its initial state with its initial variables.

§6 Nesting

A composite may use other composites from its own catalog as parts, so a status card can contain a status dot, and an export panel a date-range field.

- **Nested parts.** A part of type `self:<type>` is an instance of that type inside its parent. Its props are literals or bindings into the parent's local model, and it must be given the child's required props.
- **Events go up.** An event that a composite's machine handles in no state, whether from an action part or from an `emit`, goes to the parent's machine, and so on upwards. If no level handles it, it reaches the application as an INTENT with `source` set to the outer instance and `part` set to the dotted path, such as `dot` for the dot's `acknowledged`.
- **Parents talk down.** A transition's `signal` sends events to nested parts, as in `{"to": "busy", "signal": {"dot": "pulse"}}`, and a state's `set` may set a nested part's props. The server reaches a nested part with `signal`'s `part` path: `{"op": "signal", "id": "db01", "part": "dot", "event": "pulse"}`.
- **Values.** A nested part whose type has `value` may carry `bind`, a `/vars` pointer in the parent, and the child's value is mirrored there both ways. That is how the export panel's date range reaches its Export intent.
- **State is kept.** A nested composite keeps its state and variables when its parent changes state or hides it. It starts over only when the outer instance is opened again or a reset snapshot arrives.
- **Limits.** At most 4 levels, no type that contains itself directly or indirectly, and at most 256 parts per instance once expanded. A client treats a catalog that breaks them as unavailable, and its instances show their fallback.

```
"status-dot": {"props": {"state": {"type": "string", "enum": ["ok", "info", "warning",
"critical", "pending"], "default": "ok"}, "label": {"type": "string", "default": ""}},
  "template": [{"op": "open", "id": "s", "type": "status", "state": {"bind": "/props/
state"}, "text": {"bind": "/props/label"}},
               {"op": "open", "id": "ack", "type": "action", "label": "Acknowledge",
"intent": "ack", "hidden": true}],
  "machine": {"initial": "steady", "states": {
    "steady": {"on": {"pulse": "attention"}},
    "attention": {"set": {"s": {"text": "needs attention"}, "ack": {"hidden": false}},
                  "on": {"calm": "steady", "ack": {"to": "steady", "emit":
"acknowledged"}}, "after": {"ms": 10000, "to": "steady"}}}}}
```

The status card uses it as `{"op": "open", "id": "dot", "type": "self:status-dot", "parent": "box", "state": {"bind": "/props/status"}, "label": {"bind": "/props/status"}}`.

§7 Input, forms and intents

To the application, a composite behaves like one component.

- **Intents.** An `action` part whose `intent` no state handles sends an ordinary INTENT: `source` is the instance, the new field `part` names the part, and `context` bindings resolve against the local model. In the example, Restart sends `{"intent": "restart", "source": "web03", "part": "restart", "context": {"host": "web-03"}, …}`.
- **Form fields.** With `value`, an instance inside a form contributes `values[<instance id>]`, read from that pointer when the form is submitted. A date-range composite with `"value": "/vars"` submits `{"from": …, "to": …}`.
- **Focus and targets.** Parts take focus in template order. Input events, focus reports and scene errors name the instance as `target`; the application never addresses parts directly.

§8 Clients without the capability

The server holds every definition, so it can run a composite itself.

- **Expansion.** For a client without `composites`, the server opens the instance as a `stack` and its parts as core components with ids `<instance>.<part>`, and nested parts as `<instance>.<part>.<subpart>`, applying the props of each composite's current state.
- **Running the machine.** The server runs the machine: local events arrive as INTENTs and timers run on the server, so behaviour is the same at the cost of a round trip per event.
- **What the application sees.** Intents keep `source` set to the outer instance and `part` set to the dotted part path, exactly as from a client that runs composites.
- **Text and terminals.** Where even core components are not available, fallback rendering (v1 Section 18) shows the instance's `fallback`, then its actions as numbered menu entries.

§9 Accessibility, security and limits

A composite inherits the accessibility of its parts and adds no new attack surface.

- **Accessibility.** The instance is exposed as a group named by its `label`, with its parts' own roles inside. A state change that shows or hides parts is announced only through the parts' `live` settings, as with any component.
- **No code.** Definitions are data: no expressions, no scripts, no network access. A client interprets them within fixed limits.
- **Limits.** 256 types per catalog, 256 parts per type and per expanded instance, 4 levels of nesting, 64 props, 32 states, timers of at least 100 ms. Expanded parts count toward `scene_nodes`.
- **Emitted intents.** A client sends at most 10 emitted intents per second per instance and drops the rest, so a timer loop cannot flood the application.
- **Isolation.** Bindings reach only the instance's own model (Section 4), so one composite cannot read another's data or the region's.

§10 Wire and text-form changes

One op, two optional fields, one resource kind and one control line.

Where	Change
Capabilities	`composites`
Resource kinds	`catalog/vtx`
Region `catalogs`	A value may be a composite catalog's resource hash
Scene ops	`signal`, with an optional `part` path to a nested composite
INTENT (0x16)	`part`
Catalog content	Template parts of type `self:<type>`, and a transition's `signal` to nested parts (draft d02)
Application interface	`{"kind": "composites", "name": …, "catalog": {…}}` declares a catalog by name; a region's `catalogs` may then use the name, and the server substitutes the hash, as for themes

The bundle's `ext/composite-1/` holds `vtx-composite-1.cddl`, `vtx-composite-1.schema.json` (one catalog, whose parts the v1 open schemas check), `vtx-ops-composite-1.schema.json`, `vtx-app-composite-1.schema.json`, the Fleet catalog (a status card that nests a status dot, a date-range field, and an export panel that nests the date range), its hash vector, an application stream and a wire stream, 8 catalogs the schema rejects, 13 that break the composite rules (6 of them on nesting) and 8 instance errors. `tools/validate.py` runs 71 checks for this extension; every check in the bundle passes.