Composite Blocks in OpenHydroQual

A guide to building multi-block components

Abstract

A *composite* is a group of blocks and the links between them that the user sees and manipulates as a single block. The computational engine is unaware of composites: their members are ordinary `Block` and `Link` objects, and every solver, parameter-estimation and output path treats them individually. What a composite changes is the model file, which carries one line instead of many, and the diagram, which draws one node instead of many. This document explains how to declare a composite type in a plugin file, what the engine does with that declaration, and the pitfalls that are easy to hit and hard to diagnose.

Contents

1 Concept

A composite type is declared in an ordinary template (plugin) JSON file, next to the block and link types, and is loaded the same way with `addtemplate`. Dropping one on the diagram creates its member blocks and the links between them in a single step; the members are then driven entirely by the composite's own properties.

Three principles govern the design, and most of the practical advice in this document follows from them.

The engine never sees a composite.
Members are real blocks and links in the system's normal containers. They are assembled into the state-variable matrices exactly like anything else. A composite therefore cannot change the physics, only how a model is written down and drawn.

Members are strictly encapsulated.
A member's property values are fully determined by the composite's exposed properties. Members are never edited individually: their property panels are read-only, and any value the composite type does not expose is unreachable. If a user needs a knob the type does not offer, they *ungroup* the composite, after which it is plain blocks and links.

The composite is the unit of saving and of drawing, not of computing.
The model file carries `create composite;` with the group-level property values; the members are regenerated from the type when the file is read. In the diagram a composite is one node.

2 Anatomy of a declaration

A composite type is a top-level entry in a template JSON file with `"type": "composite"`. Beyond the usual header fields it recognises three reserved keys — `members`, `internal_links` and `external_links` — and every other entry is an ordinary quantity declaration that may carry an `applyto` mapping.

```
"Clarifier": {
  "type": "composite",
  "typecategory": "Blocks",
  "description": "Clarifier: upper and lower settling compartments",
  "icon": { "filename": "clarifier.png" },
  "helptext": "Influent and effluent attach to the upper compartment; ...",

  "members": {
    "Upper": { "type": "Settling compartment",
               "dx": "0", "dy": "0",   "_width": "200", "_height": "200" },
    "Lower": { "type": "Settling compartment",
               "dx": "0", "dy": "250", "_width": "200", "_height": "200" }
  },

  "internal_links": {
    "interface": { "type": "Sedimentation tank element interface",
                   "from": "Upper", "to": "Lower" }
  },

  "external_links": {
    "*":              { "from": "Upper", "to": "Upper" },
```

```
    "Effluent flow":  { "from": "Upper", "to": "Upper" },
    "Sludge flow":    { "from": "Lower", "to": "Lower" }
  },

  "surface_area": {
    "type": "value",
    "description": "Cross-sectional area of the compartment interface",
    "ask_user": "true", "delegate": "UnitBox", "unit": "m^2;ft^2",
    "default": "100", "estimate": "true",
    "criteria": "surface_area>0",
    "warningmessage": "The surface area must be positive!",
    "applyto": { "interface#area": "surface_area" }
  }
}
```

2.1 Header fields

Key	Meaning
`type`	Must be `"composite"`.
`typecategory`	Toolbar and object-browser grouping. `"Blocks"` puts the composite alongside ordinary blocks, which is usually what you want since the user treats it as one.
`description`	Shown as the toolbar tooltip and the menu entry.
`icon`	`{"filename": "name.png"}`, resolved against `resources/Icons`. See section 6.
`helptext`	Long-form help for the component.

2.2 Members

Each entry declares one member block. The key is the *member role* and becomes part of the instance name.

Field	Meaning
`type`	The block type to instantiate. It must exist in the metamodel when the composite is used, so make sure the template that defines it is loaded — either the same file, or one the user also adds.
`dx`, `dy`	Position *offset* from the composite's own `x`/`y`. Members are not drawn, but these offsets decide the layout after an ungroup, so give them sensible values.
`_width`, `_height`	Member node size, used after ungrouping.

2.3 Internal links

Each entry declares one link between two members. `from` and `to` are member *keys*, not instance names.

```
"internal_links": {
  "percolation_1_2": { "type": "soil_to_soil_link",
                       "from": "Soil_1", "to": "Soil_2" }
}
```

Internal links are part of the component: they are created with it, deleted with it, and never appear in the diagram. Section 7.2 covers an important obligation they carry.

2.4 External links (ports)

When a user draws a link onto a composite node, the endpoint has to be resolved to one of the members. Resolution is *by link type*:

```
"external_links": {
  "*":                        { "from": "Catchment",   "to": "Catchment" },
  "groundwater_link":         { "from": "Groundwater", "to": "Groundwater" },
  "groundwater_to_fixedhead": { "from": "Groundwater" }
}
```

The key is a link type name; `"*"` is the fallback for any type not listed explicitly. `from` applies when the composite is the link's source, `to` when it is the destination. Omitting one direction means a link of that type may not attach that way, and the user gets a clear message rather than a silent mis-connection.

Once the port resolves, the ordinary link validation runs against the *member*, so a link that requires certain quantities on its endpoints is still checked properly.

The one-type-one-port rule. A given link type can attach at only one place per direction. This is the main constraint the port design imposes, and it shapes component design. In a clarifier, influent and effluent attach to the upper compartment while return and waste sludge leave the lower one — if all four were the same generic `Fixed flow` type, they could not be told apart. The remedy is to give the distinct roles distinct link types (`Effluent flow` and `Sludge flow`), which is also self-documenting in the diagram. Declaring such a type is cheap: it is usually a copy of the generic flow link with a different name, colour and icon.

2.5 Exposed properties and mappings

Every remaining entry is an ordinary quantity declaration — the same fields you would write for a block property (`type`, `description`, `ask_user`, `delegate`, `unit`, `default`, `estimate`, `criteria`, `warningmessage`, `helptext`) — plus an optional `applyto` object that says which member quantities it drives.

```
"depth_to_groundwater": {
  "type": "value",
  "description": "Depth from the ground surface to the water table",
  "ask_user": "true", "delegate": "UnitBox", "unit": "m;ft",
  "default": "3.5", "estimate": "true",
  "applyto": {
    "Soil_1#depth": "depth_to_groundwater/7",
    "Soil_2#depth": "2*depth_to_groundwater/7",
    "Soil_3#depth": "4*depth_to_groundwater/7"
  }
}
```

Each `applyto` entry is `"<member>#<quantity>": "<mapping>"`.

- **The target** names a member block *or an internal link* — both are addressable, so an internal link's own properties can be driven from the group level.
- **The separator** between object and quantity is `#`. It is not `.` (reserved by the expression parser for the `.s`/`.e` endpoint suffixes) and not `:` (already the constituent scoping separator inside quantity names).
- **One property may drive many targets**, and many properties may drive targets on the same member. All mappings are re-applied together whenever propagation runs, so it does not matter which property hosts which mapping — group them wherever they read most clearly.

2.5.1 How a mapping is evaluated

This is the single most important rule in the document, and getting it wrong produces confusing errors.

Numeric targets — types `value`, `balance`, `constant`.
The mapping is an arithmetic *expression*, evaluated against the composite's own properties. Anything the expression engine understands is allowed.

Every other target — `source`, `timeseries`, `string`, `boolean`, `expression`.
The mapping must be the **bare name of a composite property**; its value is passed through unchanged. Arithmetic is not available here.

Note that `expression`-typed quantities fall in the second group even though they hold numbers. Several common initial conditions are declared this way — `Soil.theta` and `Groundwater cell.moisture_content` are both `expression` — so they must be driven by a bare property name:

```
"initial_moisture_content": {
  "type": "value", "ask_user": "true", "delegate": "ValueBox",
  "default": "0.2",
  "applyto": {
    "Soil_1#theta": "initial_moisture_content",
    "Soil_2#theta": "initial_moisture_content",
    "Soil_3#theta": "initial_moisture_content"
  }
}
```

If you need arithmetic on such a target — say a saturation fraction times a porosity — expose the derived value itself instead, and document the relationship in the help text.

Expressions see only composite properties. A mapping expression is evaluated against the composite, so every symbol in it must be another composite-level property. It cannot reach into a member. If a formula needs a member's parameter, expose that parameter at the group level too and drive the member from it.

2.6 Naming

Form	Example
Member instance name	`Clarifier_1__Upper` (`<instance>__<member key>`)
Internal link instance name	`Clarifier_1__interface`
Mapping target	`Upper#Storage`
Output column	`Clarifier_1__Upper_Storage` (`<block>_<quantity>`, unchanged from ordinary blocks)

Because `__` is the instance separator, avoid it in instance names you type by hand. Member keys become user-visible in the object browser and the results menu, so choose readable ones: `Upper`, `Lower`, `Soil_1`, `interface`.

3 What happens at run time

3.1 Instantiation

When a composite is created — from the toolbar, from a file, or by pasting — the following happens in order:

1. member blocks are created, named `<instance>__<key>`;
2. internal links are created and connected between them;
3. geometry is applied: each member's `x`/`y` is the composite's position plus the declared offsets, and the members' geometry quantities are hidden from the property panel;
4. constituent- and reaction-derived properties are exposed on the composite (section 3.3);
5. the group-level values from the file are assigned;
6. **propagation** runs: every `applyto` mapping is evaluated and written to its target.

Members are created in alphabetical order of their keys, so if creation order ever matters to you, name the keys accordingly.

3.2 When propagation runs

Mappings are re-applied at three moments, and it is worth knowing all three:

1. at instantiation, as above;
2. when the user edits a group-level property in the diagram or the property panel;
3. during `ApplyParameters()`, so that a calibrated value reaches the members.

The third is what makes parameter estimation work through a composite, and it has a corollary: **parameters bind at the composite level only**. A parameter assigned directly to a member quantity would be overwritten by the next propagation, so the property panel disables parameter assignment for members.

3.3 Constituents and reactions

When a constituent, reaction or reaction parameter is added, the engine copies a set of derived quantities onto every block — `X_bh:concentration` and friends. Their names are only known at run time, so a composite type cannot declare mappings for them.

They are therefore exposed automatically: for each derived quantity found on one or more members, a group-level property of the same name is created that fans out to *every* member carrying it. All members consequently share one value for such quantities. If your type declares a property with the same name explicitly, your declaration wins and no automatic mapping is generated.

4 Persistence

A composite is written as a single command carrying only its group-level properties:

```
create composite;type=Clarifier,name=Clarifier_1,x=1273,y=994,
    surface_area=100[m~^2],underflow=95[m~^3/day],upper_storage=100[m~^3], ...
```

No members or internal links appear in the file — they are regenerated from the type when it is read. External links attached to a composite are written against the *composite* name and re-resolved to a member by link type on load:

```
create link;from=Clarifier_1,to=Effluent,type=Effluent flow,flow=95,name=...
```

A consequence worth stating plainly: a model file no longer fully describes its own structure. Editing a composite type in the template changes every saved model that uses it. Treat a published composite type as an interface.

Expanded output. Both serialisers can also write a model in *expanded* form, in which composites are omitted and their members and internal links are written as ordinary blocks and links — the same output an ungrouped model produces. This is available as an API flag (`expand_composites`) for headless use and export; the GUI always saves in composite form. Note that a parameter bound to a group-level property has nowhere to live in an expanded file, so calibration setup is not preserved there.

5 Behaviour in the interface

Action	Result
Toolbar / Blocks menu	Composite types appear after the block types, with their icon and description.
Diagram	One node, drawn with the composite's icon. Members and internal links are never drawn.
Moving / resizing	Members follow, keeping their declared offsets.
Property panel	Selecting the node shows the group-level properties. Member geometry (`x`, `y`, `_width`, `_height`) is hidden, since members have no node of their own.
Object browser	The composite appears with its members and internal links nested beneath it, read-only, so member values stay inspectable.
Results menu	Grouped one submenu per member and internal link, listing that member's output items.
Ungroup	Right-click the node. Members and links become independent objects and the model saves flat afterwards. One-way: there is no regroup.
Delete	Removes the composite, its members, its internal links and any external link attached to them.
Copy / paste	Copies the composite and its internal links. External links are not copied.

6 Registering a plugin

Put the composite in any template JSON in `resources/`. To make the file selectable from the plugin chooser, add a line to `resources/default_templates.list`:

```
Hydrologic Response Unit, hydrologic_response_unit.json, hydrologic_response_unit.png
```

The three fields are the display name, the JSON file and the icon. Icons live in `resources/Icons` and follow the existing convention: 256×256 PNG with a transparent background, cropped to the artwork and centred. The same image may be used for the plugin entry and for the composite type.

7 Pitfalls

Every item below was encountered while building the two shipped composites. They share a characteristic: the model still loads and runs, so nothing draws attention to the mistake.

7.1 Drive the real initial-condition handle

Many storage quantities are derived, not set. `Soil.Storage` is declared `ask_user: false` with `initial_value_expression: "area*depth*theta"`, and `Groundwater cell.Storage` likewise de-

rives from `area*depth*moisture_content`. Mapping such a `Storage` directly appears to work — the value is written — but it is recomputed from its expression during initialisation and your value is discarded.

> **Symptom:** changing the exposed property has no effect whatsoever; two runs with different initial conditions agree to many decimal places.

Map the quantity the expression reads — `theta`, `moisture_content` — not the storage it produces. Both happen to be `expression`-typed, so by section 2.5.1 their mappings must be bare property names.

7.2 Satisfy the internal links' required properties

An internal link belongs to the composite, so nobody else will configure it. If its type declares a property with `ask_user: true` and no usable default, the composite *must* map it. `soil2groundwater_link.area` is exactly this case: no default, so it starts at zero.

> **Symptom:** a flux that should exist is identically zero — in that example, no water ever reaches the groundwater — while the model solves happily.

Before shipping a type, list every `ask_user` property of every internal link type and confirm each is either mapped or has a sound default.

7.3 Anything unexposed is unreachable

This is encapsulation working as designed, but it puts real weight on the author. A property the type does not expose cannot be edited, and cannot even be saved — it reverts to the type's default on reload. Expose everything a user might reasonably want to change, and remember that this includes initial conditions. The escape valve is ungrouping.

7.4 Watch out for confined-style head formulations

Some block types compute head in a way that punishes a plausible-looking initial condition. `Groundwater cell` uses

$$\text{head} = \text{cell top} + \frac{\text{moisture content} - \text{porosity}}{\text{specific storage}},$$

so at full saturation the head sits at the top of the cell, and a small deficit becomes a large elastic underpressure: with a specific storage of 0.01, a moisture content 0.025 below porosity puts the head 2.5 m low. Choose defaults that correspond to a sensible state, and say so in the help text.

7.5 Port ambiguity

Covered in section 2.4: one link type attaches at one place per direction. If two roles on your component need the same generic link type, introduce a distinct type for one of them.

8 A recipe

1. Decide the members and the internal links, and check that the link types you need exist and connect those block types.
2. For each internal link type, list its `ask_user` properties and plan a mapping or confirm a sound default (section 7.2).

3. Decide the ports. Group the external link types by which member they should attach to, and add a distinct link type if two roles collide (section 2.4).

4. Decide the exposed properties. Work outward from what a user would actually type: dimensions, material properties, sources, initial conditions. For each, find the member quantity that genuinely controls it (section 7.1).

5. Write the mappings, respecting the numeric / bare-name rule (section 2.5.1).

6. Give every property a description, a unit, a default, and where it helps a `criteria` with a `warningmessage`.

7. Add the icon and register the plugin (section 6).

8. Test (section 9).

9 Testing a new composite

Build a small model that uses the component and check, in order:

1. **It instantiates.** The expected number of blocks and links appear, with the expected names.

2. **Derived values are right.** Read a few member quantities back and compare against what the mappings should have produced — especially anything computed from several group properties.

3. **Every knob bites.** Change each exposed property and confirm a member value actually changes. This is what catches the initial-condition trap, and nothing else does.

4. **Ports resolve.** Draw one external link of each type and confirm it lands on the intended member.

5. **It round-trips.** Save, reload, and confirm the member values and port resolutions survive.

6. **It verifies and solves.** `VerifyAllQuantities` should report nothing, and the run should complete. Check the result is physically sensible, not merely finite.

7. **Ungroup works.** The members should survive as plain blocks with their geometry restored.

10 Worked example: a two-compartment clarifier

The full type is in `resources/wastewater.json`. Two `Settling compartment` members joined by a `Sedimentation tank element interface`; influent and effluent attach to the upper compartment, return and waste sludge leave the lower one. Because all four would otherwise be `Fixed flow`, two dedicated link types are declared alongside it, `Effluent flow` and `Sludge flow`, which give the ports something to key on and make the diagram readable. Group properties cover the interface area, the underflow, the two compartment elevations, the initial volumes, and an external inflow series.

A model using it is in `Examples/ASM_Clarifier_Composite.ohq`: an activated-sludge plant of four reactors whose settler is a single node.

11 Worked example: a hydrologic response unit

The full type is in `resources/hydrologic_response_unit.json`. Five members — a `Catchment`, three `Soil` layers and a `Groundwater cell` — with infiltration, two percolation links and a recharge link between them. It illustrates several ideas at once:

- **Derived geometry.** The user supplies a surface elevation, a depth to groundwater and a groundwater thickness; the layer thicknesses (in a $1:2:4$ ratio, finest at the surface where moisture gradients are steepest) and all four bottom elevations are computed:

```
"Soil_1#bottom_elevation": "surface_elevation-depth_to_groundwater/7",
"Soil_2#bottom_elevation": "surface_elevation-3*depth_to_groundwater/7",
"Soil_3#bottom_elevation": "surface_elevation-depth_to_groundwater"
```

- **One property, many members.** A single `area` drives the catchment, all three soil layers, the groundwater cell *and* the recharge link's interface area.
- **Selective application.** Evapotranspiration maps to the top soil layer only, while precipitation goes to the catchment.
- **Shared material properties.** The three layers discretise one soil profile rather than representing different materials, so they share one `K_sat`, one `alpha_vG` and so on. Splitting these per layer is a mechanical edit if heterogeneous layers are wanted.

A two-unit hillslope model using it is in `Examples/Hydrologic_Response_Unit/HRU_hillslope.ohq`.

12 Limitations

- **The member list is fixed.** A composite type declares a definite set of members; the count cannot be driven by a property. Variable-length structures — an n-layer column, a channel reach of n segments, an $n_x \times n_y$ aquifer grid — need one type per size, or the existing grid generator.
- **No nesting.** A composite cannot contain another composite.
- **No per-member unique data.** Values that are neither shared nor a formula over group properties — a measured conductivity for each individual layer, say — cannot be expressed. Such models are built by ungrouping.
- **Ungroup is one-way.** There is no regroup.