Phoenix Things
User Manual
Version 1.0

Table of Contents

1. Overview 3
2. Quick start 3
3. Concepts you'll use in every action 4
   3.1 Things and their parts 4
   3.2 The 12 device types 4
   3.3 The selector pattern (By / Operator / Value) 5
   3.4 Compartmentalized actions and targets 6
   3.5 Override and Overwrite toggles 6
   3.6 Picking by name or by id 6
   3.7 Connection routing summary 7
4. Actions 7
   4.1 Create Thing 8
   4.2 Edit Thing 15
   4.3 Find Thing 18
   4.4 Check for Thing 18
   4.5 Delete Thing 19
   4.6 Create Operation (upsert) 20
   4.7 Find Operation 24
   4.8 Check for Operation 24
   4.9 Delete Operation 24
   4.10 Create Rule (upsert) 26
   4.11 Find Rule 28
   4.12 Check for Rule 29
   4.13 Delete Rule 29
   4.14 Create Output 31
   4.15 Find Output 33
   4.16 Check for Output 33
   4.17 Delete Output 33
5. Field & format reference 35
   5.1 Shared enumerations 35
   5.2 Collapsed multi-line field formats 36
6. Output files 36
   6.1 <jobName>-result.json 36
   6.2 <jobName>-error.json 38
   6.3 503 retry behaviour 40
   6.4 Concurrency safety 40
7. Troubleshooting & FAQ 41

1. Overview

Phoenix Things is an Enfocus Switch flow element that lets you read from and write to your Phoenix **things library** — the library of production *devices* (presses, cutters, die-cutting tables, die-making stations) — directly from a Switch flow. Anything you can do by hand in Phoenix's things library — adding a press, editing its costing, attaching a media rule, wiring one device's output to another — you can do automatically as part of a job's journey through Switch.

A single Phoenix Things element performs any one of **17 actions**, ordered by CRUD in the dropdown: **Create** a Thing, Operation, Rule, or add an Output; **Find** or **Check for** a Thing, Operation, Rule, or Output; **Edit** a Thing; and **Delete** a Thing, Operation, Rule, or remove an Output. You pick the action from a dropdown; the form then reshapes to show only the fields that action needs.

The form is **compartmentalized**: rather than one giant "edit everything" screen, each action edits one facet of a Thing. Thing-level actions handle the device's own settings (General, Costing, Capabilities, Placement); separate actions manage its **operation modes**, its **media rules**, and its **output connections** to other devices.

Every job that leaves the element comes out one of three connections — **Success**, **Warning**, or **Error** — and (for most actions) carries a JSON file recording exactly what the element did.

> **Note:** This manual covers what the element does and how to use it. It does not explain Phoenix itself — for what a "process type", "media rule", or "operation mode" means in your imposition workflow, please refer to Phoenix's documentation.

2. Quick start

Drop a Phoenix Things element onto your flow. It needs at least one incoming connection and at least one outgoing connection. Set the connection-server properties at the top of the property panel:

Field	What it is	Default
URL	Hostname or IP address of your Phoenix server.	localhost
Port	TCP port Phoenix listens on.	8022
Interval	How often (in seconds) the element checks for and routes completed Phoenix jobs. Values below 10 are raised to 10.	10

Then choose an **Action**. The form below updates to show only the fields that action needs — so the panel looks very different for **Create Thing** than for **Create Output**. Fill the fields in, save the flow, and run a job through it.

When the job comes out **Success** (or **Warning**), look in the job's Datasets for <jobName>-result.json. When it comes out **Error**, you'll usually find <jobName>-error.json with the underlying Phoenix response.

3. Concepts you'll use in every action

3.1 Things and their parts

A **Thing** is a top-level production device. Unlike stocks, Things have **no parent** — they sit flat in the library. Each Thing owns three kinds of sub-object, and each kind has its own dedicated actions rather than being edited inside the Thing:

Part	What it is	Managed by
The Thing itself	General identity, costing, capabilities (stock/size limits), and placement. Created and edited as a whole.	Create Thing, Edit Thing, Delete Thing
Operation modes	Speed and ink-cost (or motion) settings, per named mode. Every device has one always-present "Standard" mode plus any number of named modes.	Create / Delete / Find / Check for Operation
Media rules	Per-stock overrides (marks, margins, regions, ink adjustment). Every device has one always-present "Default" rule plus any number of stock-keyed rules.	Create / Delete / Find / Check for Rule
Output connections	Directed links from this Thing to another Thing ("this press feeds that cutter").	Create Output, Delete Output, Find / Check for Output

When you **Create Thing**, Phoenix initialises the Standard operation mode, an empty media-rules set, and no connections. You then configure those with the Operation, Rule, and Output actions.

3.2 The 12 device types

Every Thing has a **type**, chosen when you create it. The type decides which fields appear and how the body is built. The 12 types are:

Type	Family	Notable extra fields
Sheet-fed Digital	Press	Sheet turn; single-pass double-sided; gripper
Web-fed Digital	Press	Single-pass double-sided; gripper (no sheet turn)
Sheet-fed Offset	Press	Setup-per-color; run-length range; minimum cost per layout; default plate; sheet position
Web-fed Offset	Press	As Sheet-fed Offset, no sheet turn
Web-fed Flexo	Press	Setup-per-color; default plate; cylinders
Flatbed Wide Format	Press	Sheet turn; gripper
Roll-fed Wide Format	Press	Gripper (no sheet turn)
Digital Cutting Table	Cutter	Estimating engine + Zund Cut Center (ZCC) settings; DCT motion operation
Guillotine Cutter	Cutter	Stack-height range; time-per-cut; no operation modes

Type	Family	Notable extra fields
Flatbed Die Cutter	Cutter	Run-length range; speed lives in costing; no operation modes
Rotary Die Cutter	Cutter	Run-length range; speed lives in costing; no operation modes
Die Making	Die	Its own costing & capabilities; no media rules, placement, or operation

The seven press types have operation modes (speed + ink cost) and press-style media rules (with regions and ink adjustment). Cutters have simpler rules and mostly no operation modes (the Digital Cutting Table is the exception — it has a motion-based operation). Die Making is the odd one out: it has no media rules, no placement, and no operation.

> **Note:** Two further types — High-Die Cutter and Corrugator — are temporarily unavailable pending a Phoenix server fix, and are not offered in the type dropdown.

3.3 The selector pattern (By / Operator / Value)

Every action **except Create Thing** picks the Thing(s) it operates on with three fields that always come together:

Field	What it does
By	The field on the Thing to look at. One of: ID, Name, External ID, Description, Notes, Type.
Operator	How to compare. One of: Equals, Not equals, Starts with, Does not start with, Contains, Does not contain, Matches (regex), Does not match.
Value	The text to compare against. Switch variables and script expressions are allowed.

Example: **By = Type, Operator = Equals, Value = SheetFedOffsetPress** selects every sheet-fed offset press. Selection is **bulk by design** — if the selector matches N Things, the action runs on all N. To act on exactly one, select **By = ID** with its GUID.

3.4 Compartmentalized actions and targets

The Operation and Rule actions first use the Thing selector (3.3) to choose which device(s) to touch, then a **target** to choose which mode or rule inside each device:

- **Operation target** — `Standard` acts on the always-present default mode; `Named` reveals a nested By / Operator / Value matcher over named modes (By ∈ Mode name, Mode ID, Stock name, Stock ID, Tool name, Tool ID — Tool applies to Digital Cutting Tables).
- **Rule target** — `Default` acts on the always-present default rule; `Named` reveals a nested matcher over stock-keyed rules (By ∈ Stock name, Stock ID).

Create Operation and **Create Rule** are **upserts**: they are keyed by the exact Mode/Stock (and, for a Digital Cutting Table operation, optionally Tool) you supply at the top of the form. A **blank** key means the always-present Standard mode / Default rule, edited in place. A non-blank key creates the named entry if absent, or — if one already exists — depends on the **Overwrite** toggle (below). Their value is always fully replaced (there is no partial merge for a mode or rule).

> **Note:** There is no separate “Edit Operation” or “Edit Rule” action — because Create Operation and Create Rule upsert, they also edit. To change an existing mode or rule, run Create with Overwrite = Yes.

3.5 Override and Overwrite toggles

Two similarly named toggles guard against clobbering data:

- **Override existing** (Create Thing) — if Yes, any existing Thing with the same name is deleted before the new one is created.
- **Overwrite** (Create Operation, Create Rule) — if a mode/rule with the same key already exists: **Yes** replaces it (an idempotent upsert); **No** (the default) fails the job to Error rather than overwriting by accident. Overwrite does not apply when the key is blank (the Standard mode / Default rule is always replaced in place).

3.6 Picking by name or by id

Phoenix Things reference each other by GUID in their JSON, but the form lets you pick by **name** from live Phoenix libraries. Where a field offers a picker (modes, stocks, sheets, plates, marks, other Things, process types, DCT tools), the accompanying **By** dropdown chooses whether your value is a name or an id. When you pick by name, the element resolves the name to an id against the Phoenix library when it builds the request; an unresolvable name is reported rather than silently succeeding.

> **Note:** The reserved mode name “Standard” never appears in a mode picker — it is reached through the Standard / blank-Mode target, not as a selectable named mode.

> **DCT tools are special:** Phoenix does not share its list of cutting tools with other programs, so the Tool picker (Digital Cutting Table operations) is filled in by the **Phoenix Scripting module** instead of a normal library. If that module isn't enabled you can still identify a tool: set Tool → By to **ID** and paste the tool's ID. See the FAQ for details.

3.7 Connection routing summary

Every job comes out one of three connections:

Connection	When
Success	The action completed as asked. The result JSON is attached (except for Check for actions).
Warning	The action completed, but did something other than what you literally asked — a matched Thing was silently *not* changed. Currently: an **Edit Thing**, **Create Operation** or **Create Rule** skipped because the Type you picked doesn't match that Thing's actual type, and a **Create Output** skipped because a Thing cannot connect to itself. The reason is in warnings.
Error	The action failed — an error reported by Phoenix, a missing required input, a reference that could not be found, an empty/invalid selector where one is required, or a **mutating action whose selector matched nothing**. A **Check for** that matches nothing also routes here (that is how Check answers "no").

If you do not draw a Warning connection, wire Warning-level jobs somewhere deliberately — an undrawn output means those jobs have nowhere to go.

4. Actions

The Action dropdown at the top of the property panel chooses one of these 17 operations:

> **Thing** — Create · Edit · Find · Check for · Delete
> **Operation** — Create · Find · Check for · Delete
> **Rule** — Create · Find · Check for · Delete
> **Output** — Create · Find · Check for · Delete

They are grouped in the dropdown by object (Thing → Operation → Rule → Output), with the CRUD verbs clustered inside each object.

Below each action, the **Properties** table lists the form fields you will see in Switch when that action is selected, and the **Result JSON** code block shows a realistic example of the file attached to the job. The two big actions (Create Thing and Edit Thing) present their fields as the shared section tables in 4.1, which every type reuses.

4.1 Create Thing

What it does. Creates a brand-new device. You give it a **Name** and a **Type**; the form then shows the General, Costing, Capabilities, and Placement sections for that type, plus any type-specific add-ons. If **Override existing = Yes** and a Thing with that name already exists, the existing Thing is deleted first.

Create is a **two-step** operation under the hood (see the FAQ for why): the element POSTs an identity skeleton so Phoenix initialises the device with its install defaults, then PUTs your full settings merged on top. You configure operation modes, media rules, and outputs afterward with their own actions.

Properties — top of the form

Field	Type	Required	Default	Notes
Name	Single-line	Yes	(blank)	The new device's name.
Type	Dropdown	Yes	Sheet-fed Digital	One of the 12 types (3.2). Decides which sections appear.
Override existing	Yes/No	No	No	If Yes, delete any existing Thing with this name first.

Properties — General (all types)

Field	Type	Required	Default	Notes
Name	Single-line	Yes	(blank)	Same as above.
Description	Single-line	No	(blank)	
Notes	Multi-line	No	(blank)	
External ID	Single-line	No	(blank)	For cross-referencing with an MIS/ERP.
Manufacturer	Single-line	No	(blank)	Omitted for Die Making.
Enforce feed type	Yes/No	No	No	Omitted for Die Making.
Allow pass-through	Yes/No	No	No	Whether jobs may pass through untouched.

> **Note:** A device's process type and feed type are hidden from the form and injected automatically from the type you pick. Change them only by editing the type defaults in the script, not in the flow.

Properties — Costing (all types except Die Making)

Field	Type	Required	Default	Notes
Rate	Single-line	No	$0.00	Cost rate.
Per	Dropdown	No	Hours	Milliseconds, Seconds, Minutes, Hours, Days.
Setup → Time	Single-line	No	0	Setup time.
Setup → Per	Dropdown	No	Hours	Milliseconds, Seconds, Minutes, Hours, Days.
Setup → Layouts	Single-line	No	0	Setup layouts.
Running waste	Single-line	No	0 %	Percentage, e.g. 10 %.

Type-specific costing add-ons: **Sheet-fed / Web-fed Offset** add *minimum cost per layout*, a *run-length range*, and a *setup-per-color* block; **Web-fed Flexo** adds the *setup-per-color* block; **Guillotine Cutter** adds *time per cut*, *average time per cut*, and *average time per turn*; **Flatbed / Rotary Die Cutter** add a *run-length range* and a *Speed* block (fixed or stepped) — for these two the run speed lives in costing, not in an operation mode. See the per-type add-on matrix at the end of this section.

Properties — Capabilities (all types except Die Making)

Field	Type	Required	Default	Notes
Min / Max width	Single-line	No	(blank)	Sheet size limits.
Min / Max height	Single-line	No	(blank)	
Min / Max caliper	Single-line	No	(blank)	
Min / Max weight	Single-line	No	(blank)	
Weight units	Dropdown	No	GSM	GSM or Lb.
Weight grade	Dropdown	No	Bond	Only when Weight units = Lb. Text, Book, Bond, Offset, Cover, Bristol, Index, Tag, Card.
Sheet handling	Dropdown	No	None	None, Long Edge Horizontal, Long Edge Vertical.
Limit allowed stocks	Dropdown	No	No limit	No limit, Inclusively, Exclusively.
Stock types	Multi-picker	No	(blank)	When limiting. Pick by name or id.
Specific stocks	Multi-picker	No	(blank)	When limiting. Pick by name or id.

Add-on: **Guillotine Cutter** adds a *stack-height* min/max here.

Properties — Placement (all types except Die Making)

Field	Type	Required	Default	Notes
Anchor	Dropdown	No	Content Area	Content Area, Sheet, Plate.
Reference point	Dropdown	No	Center	Top Left, Top Center, Top Right, Center Left, Center, Center Right, Bottom Left, Bottom Center, Bottom Right.
Placement point	Dropdown	No	Center	Top Left, Top Center, Top Right, Center Left, Center, Center Right, Bottom Left, Bottom Center, Bottom Right.
Horizontal offset (X)	Single-line	No	0"	
Vertical offset (Y)	Single-line	No	0"	
Resize sheet	Yes/No	No	No	
Default sheet	Picker	No	(blank)	Chosen from Phoenix's sheets library.

Add-ons: **Sheet-fed / Web-fed Offset** add a *Default plate* and a *Sheet position* (align X/Y + offsets) block; **Web-fed Flexo** adds a *Default plate* and a *Cylinders* list (one `Type|Circumference` per line, Type ∈ Gear or Sleeve).

Per-type add-on matrix

Type	General add-ons	Costing add-ons	Capabilities / Placement add-ons
Sheet-fed Digital	sheet turn, SPDS, gripper	—	—
Web-fed Digital	SPDS, gripper	—	—
Sheet-fed Offset	sheet turn, SPDS, gripper	min/layout, run-length, setup-per-color	default plate, sheet position
Web-fed Offset	SPDS, gripper	min/layout, run-length, setup-per-color	default plate, sheet position
Web-fed Flexo	SPDS, gripper	setup-per-color	default plate, cylinders
Flatbed Wide Format	sheet turn, SPDS, gripper	—	—
Roll-fed Wide Format	SPDS, gripper	—	—
Digital Cutting Table	estimating engine (+ ZCC)	—	—

Type	General add-ons	Costing add-ons	Capabilities / Placement add-ons
Guillotine Cutter	—	time-per-cut, avg time per cut/turn	stack-height range
Flatbed Die Cutter	—	run-length, Speed block	—
Rotary Die Cutter	—	run-length, Speed block	—
Die Making	die type	own dm-costing section	own dm-capabilities; no placement

(**SPDS** = single-pass double-sided, which reveals back-side unit counts and a double-sided speed reduction.) **Die Making** replaces the standard Costing/Capabilities with its own *dm-costing* (cost, cost-per-area, cost-per-1-up, steel-rule) and *dm-capabilities* (width/height limits, process-type/device limits), and has no Placement.

Result JSON


```
{
  "action": "Create Thing",
  "status": "ok",
  "counts": {
    "success": 2,
    "warnings": 0,
    "errors": 0,
    "failed": 0
  },
  "messages": {
    "warnings": [],
    "errors": []
  },
  "steps": [
    {
      "step": 1,
      "method": "GET",
      "endpoint": "/phoenix/libraries/things",
      "status": "ok",
      "description": "Create Thing",
      "items": [
        {
          "item": 1,
          "method": "GET",
          "endpoint": "/phoenix/libraries/things",
          "status": "ok",
          "request": null,
          "response": "…"
        }
      ]
    },
    {
      "step": 2,
      "method": "DELETE",
      "endpoint": "/phoenix/libraries/things/{thingid}",
      "status": "ok",
      "description": "Delete existing (Override)",
      "items": [
        {
          "item": 1,
          "method": "DELETE",
          "endpoint": "/phoenix/libraries/things/{thingid}",
          "status": "ok",
          "request": null,
          "response": "…"
        }
      ]
    },
    {
      "step": 3,
      "method": "POST",
      "endpoint": "/phoenix/libraries/things",
      "status": "ok",
      "description": "Create skeleton (POST)",
      "items": [
        {
          "item": 1,
```

```
        "method": "POST",
        "endpoint": "/phoenix/libraries/things",
        "status": "ok",
        "request": {
          "name": "My Press",
          "type": "SheetFedOffsetPress",
          "allow-pass-through": false,
          "connections": [],
          "properties": []
        },
        "response": {
          "resources": [
            "/libraries/things/3f9c1a20-..."
          ],
          "status-code": 201
        }
      }
    ]
  },
  {
    "step": 4,
    "method": "GET",
    "endpoint": "/phoenix/libraries/things/{thingid}",
    "status": "ok",
    "description": "Fetch Phoenix defaults (GET)",
    "items": [
      {
        "item": 1,
        "method": "GET",
        "endpoint": "/phoenix/libraries/things/{thingid}",
        "status": "ok",
        "request": null,
        "response": "…"
      }
    ]
  },
  {
    "step": 5,
    "method": "PUT",
    "endpoint": "/phoenix/libraries/things/{thingid}",
    "status": "ok",
    "description": "Apply settings (PUT)",
    "items": [
      {
        "item": 1,
        "method": "PUT",
        "endpoint": "/phoenix/libraries/things/{thingid}",
        "status": "ok",
        "request": {
          "name": "My Press",
          "id": "3f9c1a20-...",
          "version": "1",
          "costing": {},
          "capabilities": {},
          "placement": {}
        },
        "response": {
          "success": true,
          "status-code": 200
        }
```

```
        }
      ]
    }
  ],
  "result": {
    "created": [
      {
        "id": "3f9c1a20-...",
        "previousId": null,
        "name": "My Press"
      }
    ],
    "edited": [],
    "deleted": [
      {
        "id": "8ab2...",
        "previousId": null,
        "name": "My Press"
      }
    ],
    "skipped": [],
    "failed": []
  },
  "schema": {
    "app": "PhoenixThings",
    "version": "1.0"
  }
}
```


- `result.created` — the new device, as `{ id, previousId, name }`.
- `result.deleted` — what Override removed on the way in, `[]` when nothing was. **An Override delete counts:** `counts.success` is 2 here, because two devices were acted on. A plain create reports 1.
- `steps` — the stages of the create, in order. `Delete existing (Override)` only appears when Override actually deleted something, and the lookup it needs lands in the stage named after the action.
- The two-step create can be diffed stage by stage from the calls themselves: `Create skeleton (POST)` → `items[0].request` is the identity skeleton POSTed first, and `Apply settings (PUT)` → `items[0].request` is Phoenix's defaults with your settings merged on top — exactly what was saved. Each item carries Phoenix's reply in `response`.
- `messages.warnings` — always present; empty on a clean run.

4.2 Edit Thing

What it does. Updates the General / Costing / Capabilities / Placement of every matched Thing. Operation modes, media rules and connections are not touched here — edit those with their own actions.

Edit writes exactly the fields you fill in, and nothing else. Every field you leave blank keeps whatever the Thing already has; every field you fill in is written, *including* a 0, a No, or a cleared value. There is no mode to choose.

The type you pick must match the matched Thing's actual type; a mismatch is skipped and the job routes to **Warning** so you can see it happened. A selector that matches nothing routes to **Error**.

Properties

The **By / Operator / Value** selector, then the same **General / Costing / Capabilities / Placement** sections as Create Thing (4.1). Fill in only what you want to change.

Result JSON


```
{
  "action": "Edit Thing",
  "status": "ok",
  "counts": {
    "success": 1,
    "warnings": 0,
    "errors": 0,
    "failed": 0
  },
  "messages": {
    "warnings": [],
    "errors": []
  },
  "steps": [
    {
      "step": 1,
      "method": "GET",
      "endpoint": "/phoenix/libraries/things",
      "status": "ok",
      "description": "Select Things",
      "items": [
        {
          "item": 1,
          "method": "GET",
          "endpoint": "/phoenix/libraries/things",
          "status": "ok",
          "request": null,
          "response": "…"
        }
      ]
    },
    {
      "step": 2,
      "method": "GET",
      "endpoint": "/phoenix/libraries/things/{thingid}",
      "status": "ok",
      "description": "Fetch Thing (GET)",
      "items": [
        {
          "item": 1,
          "method": "GET",
          "endpoint": "/phoenix/libraries/things/{thingid}",
          "status": "ok",
          "request": null,
          "response": "…"
        }
      ]
    },
    {
      "step": 3,
      "method": null,
      "endpoint": null,
      "status": "ok",
      "description": "Write field(s)",
      "items": []
    },
    {
```

```
      "step": 4,
      "method": "PUT",
      "endpoint": "/phoenix/libraries/things/{thingid}",
      "status": "ok",
      "description": "Save Thing (PUT)",
      "items": [
        {
          "item": 1,
          "method": "PUT",
          "endpoint": "/phoenix/libraries/things/{thingid}",
          "status": "ok",
          "request": null,
          "response": "…"
        }
      ]
    }
  ],
  "result": {
    "created": [],
    "edited": [
      {
        "id": "3f9c...",
        "previousId": null,
        "name": "My Press"
      }
    ],
    "deleted": [],
    "skipped": [],
    "failed": []
  },
  "schema": {
    "app": "PhoenixThings",
    "version": "1.0"
  }
}
```


The `Write N field(s)` stage's description carries the count of individual values written — a quick way to confirm the edit did what you expected. If it reads `Write 0 field(s)`, nothing on the form was filled in. That stage makes no Phoenix call of its own, so it reports `items: []`.

4.3 Find Thing

What it does. Returns every Thing whose chosen field matches the selector, as a JSON array (empty if nothing matched). Read-only; an empty result still routes to Success. The array holds the library *summary* of each Thing (nested operation / media-rule / connection blocks are omitted — use the sub-object Find actions for those).

Properties

Field	Type	Required	Default	Notes
By	Dropdown	Yes	Name	ID, Name, External ID, Description, Notes, Type.
Operator	Dropdown	Yes	Equals	Equals, Not equals, Starts with, Does not start with, Contains, Does not contain, Matches (regex), Does not match.
Value	Single-line	No	(blank)	Text to compare against.

Result JSON


```
[
  {
    "id": "3f9c1a20-...",
    "name": "My Press",
    "type": "SheetFedOffsetPress",
    "description": "",
    "external-id": "",
    "allow-pass-through": false
  }
]
```


4.4 Check for Thing

What it does. Yes/No: does any Thing match the selector? Match → Success; no match → Error. No result file is written.

Properties

Same selector as Find Thing (By / Operator / Value).

Result JSON

None. The Yes/No outcome drives Success vs Error.

4.5 Delete Thing

What it does. Deletes every Thing the selector matches. Requires a selector — it will not delete the whole library on a blank Value.

Properties

The **By / Operator / Value** selector (bulk).

Result JSON


```
{
  "action": "Delete Thing",
  "status": "ok",
  "counts": {
    "success": 1,
    "warnings": 0,
    "errors": 0,
    "failed": 0
  },
  "messages": {
    "warnings": [],
    "errors": []
  },
  "steps": [
    {
      "step": 1,
      "method": "GET",
      "endpoint": "/phoenix/libraries/things",
      "status": "ok",
      "description": "Select Things",
      "items": [
        {
          "item": 1,
          "method": "GET",
          "endpoint": "/phoenix/libraries/things",
          "status": "ok",
          "request": null,
          "response": "…"
        }
      ]
    },
    {
      "step": 2,
      "method": "DELETE",
      "endpoint": "/phoenix/libraries/things/{thingid}",
      "status": "ok",
      "description": "Delete Thing",
      "items": [
        {
          "item": 1,
          "method": "DELETE",
          "endpoint": "/phoenix/libraries/things/{thingid}",
          "status": "ok",
```

```
          "request": null,
          "response": "…"
        }
      ]
    }
  ],
  "result": {
    "created": [],
    "edited": [],
    "deleted": [
      {
        "id": "3f9c...",
        "previousId": null,
        "name": "My Press"
      }
    ],
    "skipped": [],
    "failed": []
  },
  "schema": {
    "app": "PhoenixThings",
    "version": "1.0"
  }
}
```


4.6 Create Operation (upsert)

What it does. Adds or replaces an operation mode on every Thing the selector matches. Only presses and the Digital Cutting Table have operation modes. The mode is keyed by the **Mode** (and, for a DCT, optionally a **Stock** and/or a **Tool**) you supply at the top of the form. A **blank Mode** (or blank Mode *and* Stock *and* Tool for a DCT) targets the always-present **Standard** mode, replaced in place. A non-blank key upserts a named mode per the **Overwrite** toggle. For a DCT you must supply at least one of Stock / Mode / Tool.

Properties — key & type

Field	Type	Required	Default	Notes
Mode	Picker	No	(blank)	Blank = Standard mode. A value = a named mode.
By	Dropdown	No	Name	Whether Mode is a Name or an ID.
Overwrite	Dropdown	No	No	Yes = replace an existing same-key mode; No = fail instead.
Stock	Picker	No	(blank)	Digital Cutting Table only — part of the DCT key.
Tool	Picker	No	(blank)	Digital Cutting Table only — part of the DCT key. Selecting from the library needs the

Field	Type	Required	Default	Notes
				Phoenix Scripting module (Phoenix does not share its tool loint — see the FAQ).
Tool → By	Dropdown	No	Name	Whether Tool is a **Name** (needs the Scripting module) or a raw tool **ID** (works without the module).
Tool export folder	Single-line	No	Default	Where Phoenix writes / this app reads the tool-type list. **Default** = the app's scriptData folder (fine when Phoenix runs on the same machine as Switch). For a remote Phoenix, set a folder / UNC path both can reach. Only used when a DCT Tool is referenced by name.
Operation device type	Dropdown	No	Sheet-fed Digital	Which value shape to build: a press (speed + ink cost) or Digital Cutting Table (motion).

Properties — value (presses): Speed + Ink cost

Field	Type	Required	Default	Notes
Speed → Type	Dropdown	No	Fixed	Fixed or Stepped.
Speed → Stepped entries	Multi-line	No	(blank)	When Stepped. One `range speed` entry per line — see 5.2.
Speed → Units	Dropdown	No	Sheets	Sheets, MSI, MSF, ft, in, m, cm, mm, ft², in², m², cm², mm².
Speed → Per	Dropdown	No	Hours	Milliseconds, Seconds, Minutes, Hours, Days.
Speed → Value	Single-line	No	0	Fixed speed value.
Ink cost → Cost type	Dropdown	No	Coverage	Coverage or Clicks.
Ink cost → Ink Per	Dropdown	No	ft²	When Coverage. MSI, MSF, ft², in², m², cm², mm².
Ink cost → At	Single-line	No	1 %	Coverage percentage.
Ink cost → C / M / Y / K	Single-line	No	$0.00	Per-process ink costs.
Ink cost → Spots / Coatings / Foils	Single-line	No	$0.00	Additional ink costs.

Properties — value (Digital Cutting Table): motion

A DCT mode carries a motion block instead: lowered / lifted / lowering / lifting **velocity** and **acceleration**, a **clearing distance**, and an **auto-lift angle** and **distance**. A value of 0 in any of these tells Phoenix to use its server default for that setting.

Result JSON


```
{
  "action": "Create Operation",
  "status": "ok",
  "counts": {
    "success": 1,
    "warnings": 0,
    "errors": 0,
    "failed": 0
  },
  "messages": {
    "warnings": [],
    "errors": []
  },
  "steps": [
    {
      "step": 1,
      "method": "GET",
      "endpoint": "/phoenix/libraries/things",
      "status": "ok",
      "description": "Select Things",
      "items": [
        {
          "item": 1,
          "method": "GET",
          "endpoint": "/phoenix/libraries/things",
          "status": "ok",
          "request": null,
          "response": "…"
        }
      ]
    },
    {
      "step": 2,
      "method": "GET",
      "endpoint": "/phoenix/libraries/things/{thingid}",
      "status": "ok",
      "description": "Fetch Thing (GET)",
      "items": [
        {
          "item": 1,
          "method": "GET",
          "endpoint": "/phoenix/libraries/things/{thingid}",
          "status": "ok",
          "request": null,
          "response": "…"
        }
      ]
```

```
    },
    {
      "step": 3,
      "method": null,
      "endpoint": null,
      "status": "ok",
      "description": "Upsert named mode",
      "items": []
    },
    {
      "step": 4,
      "method": "PUT",
      "endpoint": "/phoenix/libraries/things/{thingid}",
      "status": "ok",
      "description": "Save Thing (PUT)",
      "items": [
        {
          "item": 1,
          "method": "PUT",
          "endpoint": "/phoenix/libraries/things/{thingid}",
          "status": "ok",
          "request": null,
          "response": "…"
        }
      ]
    }
  ],
  "result": {
    "created": [],
    "edited": [
      {
        "id": "3f9c...",
        "previousId": null,
        "name": "My Press"
      }
    ],
    "deleted": [],
    "skipped": [],
    "failed": []
  },
  "schema": {
    "app": "PhoenixThings",
    "version": "1.0"
  }
}
```


`result.edited` lists each device whose mode was created or replaced. A device skipped because its type didn't match, or because Overwrite = No hit an existing key, is a row in **`result.skipped`** — carrying its id and name — as well as a line in the log.

4.7 Find Operation

What it does. For every matched Thing, reports its operation modes. Led by an **Operation target**: Standard returns the default mode; Named with a blank Value returns all named modes; Named with a Value returns only the modes matching the nested By / Operator / Value (By ∈ Mode name, Mode ID, Stock name, Stock ID, **Tool name, Tool ID**). Matching by **Tool name** needs the Phoenix Scripting module (Tool ID does not). A Thing with no matching mode is omitted from the result.

Result JSON

```
[
  { "id": "3f9c...", "name": "My Press",
    "default": { "type": "OperationMode", "speed": { } } }
]
```

For a Named/all or Named/match target, each entry instead carries a matched array of the mode map entries that matched.

4.8 Check for Operation

What it does. Yes/No version of Find Operation — does any matched Thing have a matching mode? ≥1 → Success; 0 → Error. No result file.

4.9 Delete Operation

What it does. Removes named operation modes from every matched Thing. Requires **Operation target = Named** plus a non-blank Value — the always-present Standard mode cannot be deleted. The nested By / Operator / Value picks which named modes to remove (By ∈ Mode name / Mode ID / Stock name / Stock ID / Tool name / Tool ID; batch-capable, e.g. Mode name Contains “Foil”, or Tool name Equals “Cut”). Matching by Tool name needs the Phoenix Scripting module.

Result JSON


```
{
  "action": "Delete Operation",
  "status": "ok",
  "counts": {
    "success": 1,
    "warnings": 0,
    "errors": 0,
    "failed": 0
  },
  "messages": {
    "warnings": [],
    "errors": []
  },
  "steps": [
    {
      "step": 1,
      "method": "GET",
      "endpoint": "/phoenix/libraries/things",
      "status": "ok",
      "description": "Select Things",
      "items": [
        {
          "item": 1,
          "method": "GET",
          "endpoint": "/phoenix/libraries/things",
          "status": "ok",
          "request": null,
          "response": "…"
        }
      ]
    },
  ],
  "result": {
    "created": [],
    "edited": [
      {
        "id": "3f9c...",
        "previousId": null,
        "name": "My Press"
      }
    ],
    "deleted": [],
    "skipped": [],
    "failed": []
  },
  "schema": {
    "app": "PhoenixThings",
    "version": "1.0"
  }
}
```

4.10 Create Rule (upsert)

What it does. Adds or replaces a media rule on every matched Thing. The rule is keyed by **Stock**: a **blank Stock** targets the always-present **Default** rule (replaced in place); a Stock value upserts the one rule for that stock per the **Overwrite** toggle. Press rules carry regions and ink adjustment; cutter/DCT rules do not; Die Making has no media rules.

Properties — key & type

Field	Type	Required	Default	Notes
Stock	Picker	No	(blank)	Blank = Default rule. A value = a stock-keyed rule.
By	Dropdown	No	Name	Whether Stock is a Name or an ID.
Overwrite	Dropdown	No	No	Yes = replace an existing rule for that stock; No = fail instead.
Rule device type	Dropdown	No	Sheet-fed Digital	Which value shape to build (presses add regions + ink adjustment).

Properties — value

Field	Type	Required	Default	Notes
Marks	Multi-line	No	(blank)	One per line: `Mark name Side` (Side ∈ Front, Back, Both; default Both). Resolved against Phoenix's marks library.
Speed reduction	Single-line	No	0 %	Percentage.
Content margins	Single-line	No	(blank)	1, 2, or 4 pipe-separated values: 1 = all sides; 2 = Top&Bottom | Left&Right; 4 = Top|Bottom|Left|Right. Blank ⇒ 0".
Image margins	Single-line	No	(blank)	Same format as content margins.
Regions	Multi-line	No	(blank)	Presses only. One per line: Type|AlignX|AlignY|Applied|<offset/size> (offset/size takes 1, 2, or 4 values — see §5.2).
Ink adjustment	Single-line	No	0 %	Presses only. Percentage.

Result JSON


```
{
  "action": "Create Rule",
  "status": "ok",
  "counts": {
    "success": 1,
    "warnings": 0,
    "errors": 0,
    "failed": 0
  },
  "messages": {
    "warnings": [],
    "errors": []
  },
  "steps": [
    {
      "step": 1,
      "method": "GET",
      "endpoint": "/phoenix/libraries/things",
      "status": "ok",
      "description": "Select Things",
      "items": [
        {
          "item": 1,
          "method": "GET",
          "endpoint": "/phoenix/libraries/things",
          "status": "ok",
          "request": null,
          "response": "…"
        }
      ]
    },
    {
      "step": 2,
      "method": "GET",
      "endpoint": "/phoenix/libraries/things/{thingid}",
      "status": "ok",
      "description": "Fetch Thing (GET)",
      "items": [
        {
          "item": 1,
          "method": "GET",
          "endpoint": "/phoenix/libraries/things/{thingid}",
          "status": "ok",
          "request": null,
          "response": "…"
        }
      ]
    },
    {
      "step": 3,
      "method": null,
      "endpoint": null,
      "status": "ok",
      "description": "Upsert stock rule",
      "items": []
    },
    {
```

```
      "step": 4,
      "method": "PUT",
      "endpoint": "/phoenix/libraries/things/{thingid}",
      "status": "ok",
      "description": "Save Thing (PUT)",
      "items": [
        {
          "item": 1,
          "method": "PUT",
          "endpoint": "/phoenix/libraries/things/{thingid}",
          "status": "ok",
          "request": null,
          "response": "…"
        }
      ]
    }
  ],
  "result": {
    "created": [],
    "edited": [
      {
        "id": "3f9c...",
        "previousId": null,
        "name": "My Press"
      }
    ],
    "deleted": [],
    "skipped": [],
    "failed": []
  },
  "schema": {
    "app": "PhoenixThings",
    "version": "1.0"
  }
}
```


4.11 Find Rule

What it does. As Find Operation, but for media rules. Led by a **Rule target**: Default returns the default rule; Named (blank Value) returns all stock-keyed rules; Named (Value) returns the rules matching the nested Stock matcher. Result shape mirrors Find Operation (default for the default target, otherwise a matched array).

Result JSON


```
[
  { "id": "3f9c...", "name": "My Press",
    "matched": [
      { "key": { "type": "MaterialKey", "stock": { "id": "st-1" } }, "value": { } }
    ] }
]
```

4.12 Check for Rule

What it does. Yes/No version of Find Rule. ≥1 → Success; 0 → Error. No result file.

4.13 Delete Rule

What it does. Removes stock-keyed media rules from every matched Thing. Requires **Rule target = Named** plus a non-blank Value — the always-present Default rule cannot be removed. The nested Stock matcher picks which rules to remove.

Result JSON


```
{
  "action": "Delete Rule",
  "status": "ok",
  "counts": {
    "success": 1,
    "warnings": 0,
    "errors": 0,
    "failed": 0
  },
  "messages": {
    "warnings": [],
    "errors": []
  },
  "steps": [
    {
      "step": 1,
      "method": "GET",
      "endpoint": "/phoenix/libraries/things",
      "status": "ok",
      "description": "Select Things",
      "items": [
        {
          "item": 1,
          "method": "GET",
          "endpoint": "/phoenix/libraries/things",
          "status": "ok",
          "request": null,
          "response": "…"
        }
      ]
    },
    {
      "step": 2,
      "method": "GET",
      "endpoint": "/phoenix/libraries/things/{thingid}",
      "status": "ok",
      "description": "Fetch Thing (GET)",
      "items": [
        {
```

```
          "item": 1,
          "method": "GET",
          "endpoint": "/phoenix/libraries/things/{thingid}",
          "status": "ok",
          "request": null,
          "response": "…"
        }
      ]
    },
    {
      "step": 3,
      "method": null,
      "endpoint": null,
      "status": "ok",
      "description": "Remove named rule(s)",
      "items": []
    },
    {
      "step": 4,
      "method": "PUT",
      "endpoint": "/phoenix/libraries/things/{thingid}",
      "status": "ok",
      "description": "Save Thing (PUT)",
      "items": [
        {
          "item": 1,
          "method": "PUT",
          "endpoint": "/phoenix/libraries/things/{thingid}",
          "status": "ok",
          "request": null,
          "response": "…"
        }
      ]
    }
  ],
  "result": {
    "created": [],
    "edited": [
      {
        "id": "3f9c...",
        "previousId": null,
        "name": "My Press"
      }
    ],
    "deleted": [],
    "skipped": [],
    "failed": []
  },
  "schema": {
    "app": "PhoenixThings",
    "version": "1.0"
  }
}
```

4.14 Create Output

What it does. Connects every matched Thing's output to a target Thing — "this device feeds that device." The connection is de-duplicated (adding it twice is harmless), and a device is never connected to itself.

Properties

Field	Type	Required	Default	Notes
Output Thing	Picker	No	(blank)	The target device to connect to.
Output Thing by	Dropdown	No	By name	Whether the target is given by name or id.

Result JSON


```
{
  "action": "Create Output",
  "status": "ok",
  "counts": {
    "success": 1,
    "warnings": 0,
    "errors": 0,
    "failed": 0
  },
  "messages": {
    "warnings": [],
    "errors": []
  },
  "steps": [
    {
      "step": 1,
      "method": "GET",
      "endpoint": "/phoenix/libraries/things",
      "status": "ok",
      "description": "Select Things",
      "items": [
        {
          "item": 1,
          "method": "GET",
          "endpoint": "/phoenix/libraries/things",
          "status": "ok",
          "request": null,
          "response": "…"
        }
      ]
    },
    {
      "step": 2,
      "method": "GET",
      "endpoint": "/phoenix/libraries/things/{thingid}",
      "status": "ok",
```

```
      "description": "Fetch Thing (GET)",
      "items": [
        {
          "item": 1,
          "method": "GET",
          "endpoint": "/phoenix/libraries/things/{thingid}",
          "status": "ok",
          "request": null,
          "response": "…"
        }
      ]
    },
    {
      "step": 3,
      "method": null,
      "endpoint": null,
      "status": "ok",
      "description": "Add connection",
      "items": []
    },
    {
      "step": 4,
      "method": "PUT",
      "endpoint": "/phoenix/libraries/things/{thingid}",
      "status": "ok",
      "description": "Save Thing (PUT)",
      "items": [
        {
          "item": 1,
          "method": "PUT",
          "endpoint": "/phoenix/libraries/things/{thingid}",
          "status": "ok",
          "request": null,
          "response": "…"
        }
      ]
    }
  ],
  "result": {
    "created": [],
    "edited": [
      {
        "id": "3f9c...",
        "previousId": null,
        "name": "My Press"
      }
    ],
    "deleted": [],
    "skipped": [],
    "failed": []
  },
  "schema": {
    "app": "PhoenixThings",
    "version": "1.0"
  }
}
```

4.15 Find Output

What it does. Reports each matched Thing's output connections. A blank **Output Thing** reference returns all connections; a supplied reference returns only connections to that target. A Thing with no matching connection is omitted.

Result JSON


```
[
  { "id": "3f9c...", "name": "My Press",
    "matched": [ { "thing": { "id": "abc-target-id" } } ] }
]
```


4.16 Check for Output

What it does. Yes/No version of Find Output. ≥1 → Success; 0 → Error. No result file.

4.17 Delete Output

What it does. Disconnects a target Thing from every matched Thing's outputs. Give the target via **Output Thing** / **Output Thing by** (name or id).

Result JSON


```
{
  "action": "Delete Output",
  "status": "ok",
  "counts": {
    "success": 1,
    "warnings": 0,
    "errors": 0,
    "failed": 0
  },
  "messages": {
    "warnings": [],
    "errors": []
  },
  "steps": [
    {
      "step": 1,
      "method": "GET",
      "endpoint": "/phoenix/libraries/things",
      "status": "ok",
      "description": "Select Things",
      "items": [
        {
```

```
          "item": 1,
          "method": "GET",
          "endpoint": "/phoenix/libraries/things",
          "status": "ok",
          "request": null,
          "response": "…"
        }
      ]
    },
    {
      "step": 2,
      "method": "GET",
      "endpoint": "/phoenix/libraries/things/{thingid}",
      "status": "ok",
      "description": "Fetch Thing (GET)",
      "items": [
        {
          "item": 1,
          "method": "GET",
          "endpoint": "/phoenix/libraries/things/{thingid}",
          "status": "ok",
          "request": null,
          "response": "…"
        }
      ]
    },
    {
      "step": 3,
      "method": null,
      "endpoint": null,
      "status": "ok",
      "description": "Remove connection",
      "items": []
    },
    {
      "step": 4,
      "method": "PUT",
      "endpoint": "/phoenix/libraries/things/{thingid}",
      "status": "ok",
      "description": "Save Thing (PUT)",
      "items": [
        {
          "item": 1,
          "method": "PUT",
          "endpoint": "/phoenix/libraries/things/{thingid}",
          "status": "ok",
          "request": null,
          "response": "…"
        }
      ]
    }
  ],
  "result": {
    "created": [],
    "edited": [
      {
        "id": "3f9c...",
        "previousId": null,
        "name": "My Press"
      }
```

```
    ],
    "deleted": [],
    "skipped": [],
    "failed": []
  },
  "schema": {
    "app": "PhoenixThings",
    "version": "1.0"
  }
}
```

5. Field & format reference

Shared enumerations and the line formats used by the collapsed multi-line fields.

5.1 Shared enumerations

Enum	Values
Time units (Per)	Milliseconds, Seconds, Minutes, Hours, Days
Speed units	Sheets, MSI, MSF, ft, in, m, cm, mm, ft², in², m², cm², mm²
Area units (ink cost / dm cost per area)	MSI, MSF, ft², in², m², cm², mm²
Nine-point (anchor / reference / placement)	Top Left, Top Center, Top Right, Center Left, Center, Center Right, Bottom Left, Bottom Center, Bottom Right
Weight grades (when Lb)	Text, Book, Bond, Offset, Cover, Bristol, Index, Tag, Card
Comparators	Equals, Not equals, Starts with, Does not start with, Contains, Does not contain, Matches (regex), Does not match
Operation matcher By (Find/Delete Operation)	Mode name, Mode ID, Stock name, Stock ID, Tool name, Tool ID

5.2 Collapsed multi-line field formats

Several repeating groups are entered as a single multi-line field, one entry per line, with columns separated by a pipe | (never a comma — values such as $1,000.00 contain commas). Malformed lines are skipped with a log warning; missing trailing columns fall back to defaults; blank lines and None are ignored.

Group	Line format
Marks	`Mark name Side` — Side ∈ Front, Back, Both (default Both).
Regions	Type|AlignX|AlignY|Applied|<offset/size>. The trailing offset/size block takes **1** value (all four), **2** values (Offset X&Y | Size X&Y), or **4** values (OffsetX|OffsetY|SizeX|SizeY); blank ⇒ 0". (e.g. Rectangle|Center|Center|Always|0.5" · …|Always|0"|0.5" · …|Always|0"|0"|0.5"|0.5".)
Margins (Content / Image)	1, 2, or 4 pipe-separated values: **1** = all four sides; **2** = Top&Bottom | Left&Right; **4** = Top|Bottom|Left|Right. Blank sides default 0". (e.g. 0.125" · 0.125"|0.25" · 0.125"|0.125"|0"|0".)
Cylinders (Flexo)	`Type Circumference` — Type ∈ Gear, Sleeve.
Stepped speed	`range speed — the run-length threshold, then the rate applied at and beyond it. Both accept decimals, and a thousands separator is fine (1,000 1,200`).

6. Output files

6.1 <jobName>-result.json

Written when the action completed.

- **Find** actions produce a JSON array at the top level ([] if nothing matched).
- **Check for** actions write no file — the Yes/No outcome drives Success vs Error only.
- **Create / Edit / Delete** actions produce the shared envelope — the same shape used by Phoenix Stocks, Plates and Dies, so one downstream script can read all four:

Key	What it holds
action	Which action produced this.
status	ok, warning or error — the job's overall outcome, matching the connection it took.
counts	{ success, warnings, errors, failed }, all derived. See below.
messages	{ warnings, errors } — why the job routed to Warning or Error, each entry pinned to a stage.
steps	The stages the action ran, in order, each with the calls it made. Never empty — see §6.1.1.

Key	What it holds
result	The five outcome buckets. Always all five, always arrays.
schema	{ app, version } — which element produced this, and the envelope version (currently "1.0").

result — the five buckets. created, edited, deleted, skipped, failed. The bucket a row sits in **is** what happened to it, so there is no op field to read. Every row is { id, previousId, name } and nothing else; a failed row adds error. For the sub-object actions (Operation / Rule / Output), edited lists the Things whose sub-object was changed. failed is always [] for Things — but it is always present, so one script reads all four elements the same way.

skipped is the one to know about. A Thing that matched your selector and was deliberately left alone lands there: a wire type that does not match the type you chose, an operation block of the wrong container type, a Delete that found no named mode to remove. Those used to be visible only as a sentence in messages.warnings; now they are countable rows carrying the Thing's id and name.

counts. success is created + edited + deleted — the number of Things successfully written. A **skip is not a success**, and skipped has no count of its own, so success + failed does not necessarily cover every row; read result.skipped.length. warnings and errors count *messages*, not Things. A Create Thing with Override reports success: 2, because it deleted one Thing and created another.

6.1.1 Reading steps

Several actions do more than one thing — Create Thing makes two Phoenix calls, and every sub-object action fetches each matched Thing before writing it back. steps reports those stages in the order they ran:

Field	What it holds
step	Stage number, starting at 1.
description	What the stage did, e.g. Create skeleton (POST).
status	ok, warning (something diverged from what you asked), or error.
method / endpoint	The HTTP verb and path of the stage's **first** call — or null when it made none.
items	**One entry per HTTP call the stage made.** See below.

Each items entry is { item, method, endpoint, status, request, response }: the call's own verb and path, whether it succeeded, the body that was sent (null for a GET or DELETE), and Phoenix's reply. So you can diff what went out against what came back without the element having to echo either into a separate key — which is what the old per-stage data object was for.

Four things to know when reading it:

- **A stage that did not run is absent, not listed as skipped.** If you turned Override on but nothing matched, there is no Delete existing (Override) stage — its presence is how you tell a replacement from a plain create.
- **items.length is the batch size** for a per-Thing stage — one call per Thing that reached it, which can be lower than the match count if some were skipped. A one-off stage like Select Things has exactly one.
- **A stage that makes no HTTP call reports items: [] and a null method.** The stages that only rearrange data — Write field(s), Upsert named mode, Add connection — are like this. They are still worth reading: they name the domain step, and a warning raised in one is attributed to it.
- **A call made before the first named stage lands in a stage named after the action.** Create Thing's Override lookup does this, which is why the first stage of that sample reads Create Thing. Every call belongs to exactly one stage.

steps is always present and never empty — a single-stage action reports one stage named after the action,
so a script can loop over it without special-casing.

> In the samples above, a response shown as "…" is elided for brevity — in a real result it is whatever Phoenix actually returned.

6.2 <jobName>-error.json

Written when Phoenix reported an error, or when the retries described below were exhausted. Validation errors from the element itself (missing required input, unresolvable key, blank selector where one is required) route to Error with a log message but no error file.

It is the same envelope as the result file — the same seven keys, in the same order, so one downstream script can parse either branch. messages.errors explains what failed and names the stage and call it happened on, and steps shows how far the action got: the stage that failed is marked "error", and **Phoenix's own error body is the failing call's response**.

To find what went wrong: scan steps[].items[] for status: "error". That item's response is Phoenix's raw reply and its request is exactly what was rejected, while messages.errors carries the readable version with the same step/item coordinates:


```
{
  "action": "Create Thing",
  "status": "error",
  "counts": {
    "success": 0,
    "warnings": 0,
    "errors": 1,
    "failed": 0
  },
  "messages": {
```

```
    "warnings": [],
    "errors": [
      {
        "step": 3,
        "item": 1,
        "message": "HTTP 400: An error occurred while trying to save X item"
      }
    ]
  },
  "steps": [
    {
      "step": 1,
      "method": "POST",
      "endpoint": "/phoenix/libraries/things",
      "status": "ok",
      "description": "Create skeleton (POST)",
      "items": [
        {
          "item": 1,
          "method": "POST",
          "endpoint": "/phoenix/libraries/things",
          "status": "ok",
          "request": null,
          "response": "…"
        }
      ]
    },
    {
      "step": 2,
      "method": "GET",
      "endpoint": "/phoenix/libraries/things/{thingid}",
      "status": "ok",
      "description": "Fetch Phoenix defaults (GET)",
      "items": [
        {
          "item": 1,
          "method": "GET",
          "endpoint": "/phoenix/libraries/things/{thingid}",
          "status": "ok",
          "request": null,
          "response": "…"
        }
      ]
    },
    {
      "step": 3,
      "method": "PUT",
      "endpoint": "/phoenix/libraries/things/{thingid}",
      "status": "error",
      "description": "Apply settings (PUT)",
      "items": [
        {
          "item": 1,
          "method": "PUT",
          "endpoint": "/phoenix/libraries/things/{thingid}",
          "status": "error",
          "request": null,
          "response": {
            "success": false,
            "status-code": 400,
```

```
            "errors": [
              {
                "id": 400,
                "text": "An error occurred while trying to save X item"
              }
            ],
            "warnings": [],
            "resources": []
          }
        }
      ]
    }
  ],
  "result": {
    "created": [],
    "edited": [],
    "deleted": [],
    "skipped": [],
    "failed": []
  },
  "schema": {
    "app": "PhoenixThings",
    "version": "1.0"
  }
}
```


> All five `result` buckets are `[]` on this branch even when the action had already written something
> before it failed — read `steps` to see how far it got. In the example above the device *was* created by
> the `Create skeleton (POST)` stage; only your settings failed to apply.

6.3 503 retry behaviour

When Phoenix reports HTTP 503 (“at maximum concurrency”), the element retries automatically up to 5 times with an increasing delay (about 10, 15, 20, 25 seconds) before giving up and routing to Error.

6.4 Concurrency safety

All four Phoenix library apps (Things, Stocks, Dies, Plates) serialize their writes through one shared on-disk lock, so only one mutation touches the Phoenix library at a time — concurrent writes can corrupt Phoenix’s data. Read-only Find / Check for actions don’t take the lock and run concurrently. No configuration is required; the lock recovers automatically after a crash.

7. Troubleshooting & FAQ

The form looks completely different from one action to the next.

That is intentional. The panel is compartmentalized — each action shows only the fields it needs, so Create Thing exposes the full device form while Create Output shows just a target picker.

My Check for … came out the Error port.

A Check for action that matches nothing routes to Error on purpose — that is how it answers “no.” Use the matching Find action if you want the data without Error routing.

Why does Create Thing make two calls?

Phoenix will not accept a complete device in one go for most types. So the element creates the device with just its name and type first (Phoenix accepts that and fills in its own defaults), reads it back, then saves your settings on top. The result file’s steps array shows every stage — Create skeleton (POST), Fetch Phoenix defaults (GET) and Apply settings (PUT) — each with the body it sent and Phoenix’s reply.

My Edit Thing didn’t change something.

Edit writes exactly the fields you fill in. A field left blank is not written at all, so the Thing keeps whatever it had. A 0, a No, or a cleared value **is** written — those are real values, not blanks. Check the Write N field(s) stage in the result JSON's steps: the count in its description is how many values were written, so Write 0 field(s) means nothing on the form was filled in.

If a *matched* Thing was skipped, the job routes to **Warning**, the Thing appears in result.skipped, and the reason is in messages.warnings — the usual cause is that the **Type** you picked does not match that Thing’s actual type.

I edited a mode/rule but it didn’t change.

Create Operation / Create Rule refuse to overwrite an existing key unless **Overwrite = Yes**. Set Overwrite to Yes to replace an existing mode or rule.

How do I key a Digital Cutting Table operation by Tool?

DCT operations can be keyed by **Stock, Mode, and/or Tool** (at least one). On **Create Operation**, fill the **Tool** field (with **Stock** and **Operation device type = Digital Cutting Table**); on **Find / Delete Operation**, set the matcher **By** to **Tool name** or **Tool ID**.

Why does the Tool picker need the Phoenix Scripting module?

Phoenix does not share its list of cutting tools with other programs (unlike stocks or modes), and a tool can only be identified by its ID. To let you pick a tool by **name**, the element uses the **Phoenix Scripting module** to read the tool list from inside Phoenix. If that module isn't enabled, the picker will be empty and using a tool **name** fails the job with a clear message — but you can always set **Tool → By = ID** and paste the tool's ID, which needs no Scripting module.

The Tool picker is empty / "tool not found" even though the tool exists.

Three things to check: (1) the **Phoenix Scripting module** is licensed and enabled; (2) Phoenix and Switch can reach the same folder — the tool list is handed over as a file, so for a **remote** Phoenix you must set **Tool export folder** to a shared folder / UNC path both machines can access (leave it **Default** when Phoenix runs on the same machine as Switch); (3) the name matches exactly (or use **By = ID** with the tool's ID). Nothing is written to your Phoenix data by listing tools — the element creates and immediately closes a temporary project to run the lookup.

Where did High-Die Cutter and Corrugator go?

They are temporarily withheld pending a Phoenix server fix and will return in a later version.

Can I use Switch variables in the fields?

Yes — text fields accept Switch variables and inline script expressions, including selector Values and the collapsed multi-line fields, so the element can be fully data-driven.