Phoenix Plates
User Manual
Version 1.0

Table of Contents

1. Overview

Phoenix Plates is an Enfocus Switch flow element that lets you read from and write to your Phoenix **plates library** directly from a Switch flow. A *plate* is a print-plate asset in the Phoenix Imposition library — a flat record describing the plate's geometry (width, height, punch height), its press-distortion compensation (horizontal and vertical), and its identity (name, external ID, description) and price. Anything you can do by hand in Phoenix's plates library — adding a plate, editing its dimensions, deleting an old one — you can do automatically as part of a job's journey through Switch.

A single Phoenix Plates element performs any one of **5 actions**: **Find**, **Check for**, **Create**, **Edit**, or **Delete** a plate. You pick the action from a dropdown in the element's property panel; the rest of the form changes to show only the fields that action needs.

Every job that leaves the element comes out one of two connections — **Success** or **Error** — and (for most actions) carries a JSON file recording exactly what the element did. You can wire your flow off either output and read the JSON in a downstream script for richer logic.

> **Note:** This manual covers what the element does and how to use it. It does not explain Phoenix itself — for what plate distortion means or how plates are used in imposition, please refer to Phoenix's documentation.

2. Quick start

Drop a Phoenix Plates 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** from the dropdown. The form below updates to show only the fields that action needs. Fill them in, save the flow, and run a job through it.

When the job comes out the **Success** port, look in the job's Datasets — you'll find a file called `<jobName>-result.json` with the structured result. When it comes out the **Error** path, you'll usually find `<jobName>-error.json` with the underlying Phoenix response (when Phoenix itself reported an error).

3. Concepts you'll use in every action

3.1 The plate

Unlike stocks, plates are **flat** — there is no tree, no parent, no sub-objects. A plate is a single record with these fields:

Field	Type	Notes
id	string	Server-assigned GUID. Read-only — you never set it; you receive it back on Create and in Find results.
name	string	The plate's unique name in the library. Required on Create. Two plates cannot share a name.
external-id	string	Optional identifier for cross-referencing with an MIS or ERP.
description	string	Free-text description shown in Phoenix lists. Phoenix can only write this when a plate is created — see 4.4.
width	string	Plate width. Unit-bearing string, e.g. `23in` or `584mm`. Required on Create.
height	string	Plate height. Unit-bearing string, e.g. `29in` or `737mm`. Required on Create.
punch-height	string	Punch height. Unit-bearing string, e.g. `0.5in` or `12.7mm`.
h-distortion	number	Horizontal distortion. You enter a **percentage** (`100` = no distortion); Phoenix stores it as a multiplier, so `99.8` is saved as `0.998`.
v-distortion	number	Vertical distortion. Same percentage convention as above.
price	number	Price of one plate, a decimal, e.g. `12.50`.

Every action targets a plate (or a set of plates) at this single level.

Name, Width and Height are all required to create a plate. Phoenix refuses to store a plate that is missing any of the three, and answers with an unhelpful *"An error occurred while trying to save Plate item with ID …"*. The element checks for them first and fails the job with a message naming the missing field instead.

Distortion is a percentage. `100` means no distortion, `99.8` means a 0.2% shrink. A trailing `%` is accepted. Because a raw multiplier such as `1.0` typed into a percentage field would otherwise be saved as a 100× shrink, any value below `50` is rejected with an explanatory error rather than applied.

3.2 The selector pattern (By / Operator / Value)

To pick which plate(s) an action operates on, the form uses three fields that always come together:

Field	What it does
By	The field on the plate to look at. One of: ID, Name, External ID, Description.
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 = Name, Operator = Starts with, Value = 23x29** selects every plate whose name begins with “23x29”.

Filtering happens **client-side**: the element fetches the whole plates library from Phoenix and matches in the flow. A couple of edge cases worth knowing:

- A **blank Value** with Equals matches only plates whose field is literally empty (usually none). A blank Value with Contains matches *every* plate (everything “contains” the empty string).
- Matches / Does not match treat the Value as a regular expression. Invalid or pathologically long patterns simply match nothing — they never crash the job.

The selector appears for **Find, Check for, Edit, and Delete**. **Create** has no selector — it always makes a new plate.

3.3 Bulk operations

Edit and **Delete** act on **every** plate the selector matches, not just the first. If **By = Description, Operator = Contains, Value = offset** matches 8 plates, all 8 are edited (or deleted). This is by design, so you can build sweeping cleanup or re-pricing flows in one element. To act on exactly one plate, select **By = ID** with its GUID — IDs are unique.

3.4 The “Override existing” toggle

Create Plate has an **Override existing** Yes/No field. When set to **Yes**, the element first deletes any existing plate with the same **Name**, then creates the new one — an idempotent “make sure this plate exists exactly as described” operation. When **No** (the default), and Phoenix rejects a duplicate name, the create fails to the Error port. When Override actually deletes something, the deleted plate’s id is recorded in the result JSON as overrideDeletedId.

3.5 Connection routing summary

Phoenix Plates uses three-light **Success / Warning / Error** routing:

Connection	When
Success	The action completed as asked. For Find, the matches (possibly an empty array) are attached as <jobName>-result.json.
Warning	The action completed, but did something other than what you literally asked. Currently one case: a Description change forced a re-create and the **old copy could not be deleted**, so it is still in the library under a temporary name and needs removing by hand.
Error	The action failed — an error reported by Phoenix, a missing required input, a Delete with a blank Value, or an **Edit or Delete whose selector matched nothing**. A **Check for Plate** 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 5 operations:

> Find Plate · Check for Plate · Create Plate · Edit Plate · Delete Plate

Below each action, the **Properties** table lists every form field you'll see in Switch when that action is selected, in the order it appears in the panel. The **Result JSON** code block shows a realistic example of the file attached to the job.

4.1 Find Plate

What it does. Returns every plate whose chosen field matches the selector. The result is a JSON array — empty if nothing matched. Find is read-only; nothing in Phoenix is changed, and an empty result still routes to **Success**.

Properties

Field	Type	Required	Default	Notes
By	Dropdown	Yes	Name	ID, Name, External ID, Description.
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

A JSON array of matched plate records:

```
[
  {
    "id": "550e8400-e29b-41d4-a716-446655440000",
    "name": "23x29 Standard Plate",
    "external-id": "PLATE-001",
    "description": "Standard 23\" x 29\" offset plate",
    "width": "584mm",
    "height": "737mm",
    "punch-height": "12.7mm",
    "h-distortion": 0.0015,
    "v-distortion": 0.0015,
    "price": 12.5
  }
]
```

If nothing matches, the result is `[]` and the job still routes to Success.

4.2 Check for Plate

What it does. Tells you, with a Yes/No outcome, whether any plate matches the selector. It runs the same filter as Find and asks “did I get at least one result?”

- **Match found** → job goes out the **Success** port. No result file is written; the count is logged to the job log.
- **No match** → job goes out the **Error** port. This is by design — it makes “is this plate missing?” trivial to branch on.

Properties

Field	Type	Required	Default	Notes
By	Dropdown	Yes	Name	ID, Name, External ID, Description.
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

None. Check actions don’t attach a result file — the Yes/No outcome drives Success vs Error only.

4.3 Create Plate

What it does. Creates a brand-new plate in Phoenix in one step. **Name**, **Width** and **Height** are required; every other field is optional and omitted from the request when left blank. If **Override existing = Yes** and a plate with that name already exists, the existing plate is deleted first.

Every field except Name, External ID and Description arrives pre-filled: Width and Height at 1", Punch height at 0", both distortions at 100% and Price at 0.00. Overwrite them as needed. Clearing an optional one simply omits it from the request, and Phoenix applies the same value itself; clearing Width or Height fails the job, since Phoenix requires them.

Name has no default because plate names must be unique, and External ID and Description are left blank so an unconfigured element never writes placeholder text into your library.

Properties

Field	Type	Required	Default	Notes
Name	Single-line	Yes	(blank)	The new plate’s unique name. Example: 23x29 Standard Plate.
Override existing	Yes/No	No	No	If Yes, delete any existing plate with this name before creating.
External ID	Single-line	No	(blank)	For cross-referencing with an MIS or ERP.
Description	Single-line	No	(blank)	Asset description shown in Phoenix lists.
Width	Single-line	**Yes**	1"	Unit-bearing. Example: 23in or 584mm.
Height	Single-line	**Yes**	1"	Unit-bearing. Example: 29in or 737mm.
Punch height	Single-line	No	0"	Unit-bearing. Example: 0.5in or 12.7mm.
Horizontal distortion	Single-line	No	100	Percentage; 100 = no distortion. Example: 99.8 for a 0.2% shrink.
Vertical distortion	Single-line	No	100	Percentage; 100 = no distortion. Example: 99.8 for a 0.2% shrink.
Price	Single-line	No	0.00	Decimal price of one plate. Example: 12.50.

Result JSON


```
  {
    "action": "Create Plate",
    "status": "warning",
    "counts": { "success": 2, "warnings": 1, "errors": 0, "failed": 0 },
    "messages": {
      "warnings": [
        { "step": 2, "item": 1,
          "message": "Create Plate — Override deleted the existing \"23x29 Standard Plate\"
(id=550e8400-e29b-41d4-a716-446655440099) before creating the new one." }
      ],
      "errors": []
    },
    "steps": [
      { "step": 1, "method": "GET", "endpoint": "/phoenix/libraries/plates",
        "status": "ok", "description": "Create Plate",
        "items": [
          { "item": 1, "method": "GET", "endpoint": "/phoenix/libraries/plates",
            "status": "ok", "request": null,
            "response": [ { "id": "550e8400-e29b-41d4-a716-446655440099", "name": "23x29
Standard Plate" } ] }
        ] },
      { "step": 2, "method": "DELETE", "endpoint": "/phoenix/libraries/plates/{plateid}",
        "status": "warning", "description": "Delete existing (Override)",
        "items": [
          { "item": 1, "method": "DELETE", "endpoint": "/phoenix/libraries/plates/{plateid}",
            "status": "ok", "request": null, "response": null }
        ] },
      { "step": 3, "method": "POST", "endpoint": "/phoenix/libraries/plates",
        "status": "ok", "description": "Create plate (POST)",
        "items": [
          { "item": 1, "method": "POST", "endpoint": "/phoenix/libraries/plates",
            "status": "ok",
            "request": { "name": "23x29 Standard Plate", "width": "584mm", "height": "737mm",
"price": 12.5 },
            "response": { "resources": ["/phoenix/libraries/plates/550e8400-e29b-41d4-a716-
446655440001"] } }
        ] }
    ],
    "result": {
      "created": [ { "id": "550e8400-e29b-41d4-a716-446655440001", "previousId": null, "name":
"23x29 Standard Plate" } ],
      "edited":  [],
      "deleted": [ { "id": "550e8400-e29b-41d4-a716-446655440099", "previousId": null, "name":
"23x29 Standard Plate" } ],
      "skipped": [],
      "failed":  []
    },
    "schema": { "app": "PhoenixPlates", "version": "1.0" }
  }
```


The example above has Override **on** and a name clash, which is the busiest a Create gets. A plain create has one step, `deleted` empty, and `status: "ok"`.

- result.created — the new plate, as { id, previousId, name }. previousId is null on a create.
- result.deleted — what Override removed on the way in. **Override deleted a real plate, so it counts:** counts.success is 2 here, because two plates were successfully acted on. A plain create reports 1.
- steps — the stages of the run, each holding one entry per HTTP call it made. Delete existing (Override) only appears when Override actually removed a plate.
- steps[2].items[0].request is exactly the fields the element sent (blank fields are omitted), and .response is Phoenix's reply — so you can diff the two without the element having to echo either.
- The library lookup that Override needs runs before any named stage, so it lands in step 1 under the action's own name. That is how *every* call ends up belonging to exactly one stage.

4.4 Edit Plate

What it does. Updates every plate the selector matches. Edit is a **partial update**: the element fetches each matched plate, overlays only the fields you filled in, and saves it back. Fields left blank keep their existing value. Leaving **Name** blank keeps each plate's existing name.

> **Note:** There is no "full replace" mode. Because blank fields are treated as "leave unchanged," you cannot use Edit to clear a value back to empty — set it to a new value, or delete and recreate the plate.

The Edit fields are separate from the Create fields and all start blank, so an edit only ever touches what you actually typed. Setting Price to 0 is a real edit and is saved as 0.00; leaving Price blank keeps whatever the plate already had.

Changing the Description re-creates the plate

Phoenix cannot update a plate's description in place. It accepts the update, reports success, and leaves the stored description unchanged — the field can only be written when a plate is created. So when the Description you enter differs from the one on a matched plate, the element re-creates that plate instead:

1. the existing record is parked under a temporary name,
2. a replacement is created under the real name, carrying the new description and every other value,
3. the old record is deleted.

The plate keeps its name, its dimensions, its distortion, its price and its external ID — but it is issued a **new id**. If anything in Phoenix refers to that plate by id rather than by name, update it afterwards. The element logs both ids (old → new) and reports them in the result JSON. If the replacement cannot be created, the original name is restored and the plate is left exactly as it was.

Editing any other field never re-creates anything, and neither does re-entering a Description that already matches.

Properties

Field	Type	Required	Default	Notes
By	Dropdown	Yes	Name	Selector field: ID, Name, External ID, Description.
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)	Which plate(s) to edit. Matches in bulk.
Name	Single-line	No	(blank)	New name. Blank = keep existing.
External ID	Single-line	No	(blank)	Blank = keep existing.
Description	Single-line	No	(blank)	Blank = keep existing. A change re-creates the plate under a new id — see above.
Width	Single-line	No	(blank)	Unit-bearing. Blank = keep existing.
Height	Single-line	No	(blank)	Unit-bearing. Blank = keep existing.
Punch height	Single-line	No	(blank)	Unit-bearing. Blank = keep existing.
Horizontal distortion	Single-line	No	(blank)	Percentage; 100 = no distortion. Blank = keep existing.
Vertical distortion	Single-line	No	(blank)	Percentage; 100 = no distortion. Blank = keep existing.
Price	Single-line	No	(blank)	Decimal. Blank = keep existing. 0 is a real value, not a blank.

Result JSON


```
{
  "action": "Edit Plate",
  "status": "ok",
  "counts": { "success": 2, "warnings": 0, "errors": 0, "failed": 0 },
  "messages": { "warnings": [], "errors": [] },
  "steps": [
    { "step": 1, "method": "GET", "endpoint": "/phoenix/libraries/plates",
      "status": "ok", "description": "Select plates", "items": [ "…1 call" ] },
    { "step": 2, "method": "GET", "endpoint": "/phoenix/libraries/plates/{plateid}",
      "status": "ok", "description": "Fetch plate (GET)", "items": [ "…2 calls" ] },
    { "step": 3, "method": "PUT", "endpoint": "/phoenix/libraries/plates/{plateid}",
      "status": "ok", "description": "Rename original aside (PUT)", "items": [ "…1 call" ] },
    { "step": 4, "method": "POST", "endpoint": "/phoenix/libraries/plates",
      "status": "ok", "description": "Create replacement (POST)", "items": [ "…1 call" ] },
    { "step": 5, "method": "DELETE", "endpoint": "/phoenix/libraries/plates/{plateid}",
      "status": "ok", "description": "Delete original (DELETE)", "items": [ "…1 call" ] }
  ],
  "result": {
    "created": [],
    "edited": [
      { "id": "550e8400-e29b-41d4-a716-446655440000", "previousId": null,
        "name": "23x29 Standard Plate" },
      { "id": "550e8400-e29b-41d4-a716-4466554400ff",
        "previousId": "550e8400-e29b-41d4-a716-446655440001",
        "name": "23x29 Wide Plate" }
    ],
    "deleted": [],
    "skipped": [],
    "failed":  []
  },
  "schema": { "app": "PhoenixPlates", "version": "1.0" }
}
```


(items is abbreviated above; each entry holds { item, method, endpoint, status, request, response } as shown in full under Create Plate.)

result.edited lists every plate updated, and counts.success is its length.

- **previousId is how you tell an in-place edit from a re-create.** It is null when the plate kept its
  id, and holds the **old** id when the Description change forced a delete-and-recreate. It is always
  present, so a downstream step can read it unconditionally instead of testing a flag first. The two rows
  above are one of each.
- The two plates took different paths, which is why steps 3–5 exist at all. Each stage reports one item
  per call it made, so a batch of five in-place edits gives step 2 five items and no steps 3–5.

- What Phoenix returned for each save is `steps[n].items[i].response` — `null` on an in-place PUT, since
  Phoenix answers 204 No Content, and the create reply on a re-create.

A selector that matches nothing routes to **Error**, not Success — an empty edit reported as success
is indistinguishable from one that actually changed something.

4.5 Delete Plate

What it does. Deletes every plate the selector matches. To guard against wiping the whole library by accident, Delete **requires a non-blank Value** — a blank Value fails the job to Error rather than deleting everything.

Properties

Field	Type	Required	Default	Notes
By	Dropdown	Yes	Name	ID, Name, External ID, Description.
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	Yes	(blank)	Required — no bare delete-all. Matches in bulk.

Result JSON


```
  {
    "action": "Delete Plate",
    "status": "ok",
    "counts": { "success": 1, "warnings": 0, "errors": 0, "failed": 0 },
    "messages": { "warnings": [], "errors": [] },
    "steps": [
      { "step": 1, "method": "GET", "endpoint": "/phoenix/libraries/plates",
        "status": "ok", "description": "Select plates",
        "items": [
          { "item": 1, "method": "GET", "endpoint": "/phoenix/libraries/plates",
            "status": "ok", "request": null,
            "response": [ { "id": "550e8400-e29b-41d4-a716-446655440000", "name": "23x29
Standard Plate" } ] }
        ] },
      { "step": 2, "method": "DELETE", "endpoint": "/phoenix/libraries/plates/{plateid}",
        "status": "ok", "description": "Delete plate",
        "items": [
          { "item": 1, "method": "DELETE", "endpoint": "/phoenix/libraries/plates/{plateid}",
            "status": "ok", "request": null, "response": null }
        ] }
    ],
    "result": {
      "created": [],
      "edited":  [],
      "deleted": [ { "id": "550e8400-e29b-41d4-a716-446655440000", "previousId": null, "name":
"23x29 Standard Plate" } ],
      "skipped": [],
      "failed":  []
    },
    "schema": { "app": "PhoenixPlates", "version": "1.0" }
  }
```


Each deleted plate is one row in result.deleted, and one item on the Delete plate stage — so a batch
of twelve gives twelve rows and twelve items. The item response is null because Phoenix returns 204 No
Content on delete. If the selector matches nothing, the job routes to Error.

5. Output files

Every job that passes through Phoenix Plates may attach one JSON file as a Switch child log. Which one (if any) appears depends on how the job left the element.

5.1 <jobName>-result.json

Written when the action completed.

- For **Find**, the file is a JSON array at the top level ([] if nothing matched).
- For **Check for**, no file is written — the Yes/No outcome drives Success vs Error only.
- For **Create / Edit / Delete**, the file is the shared envelope — the same shape used by Phoenix Stocks, Things 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 the note 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.
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. Because a row is always the same three keys, a script can read previousId unconditionally rather than testing whether a re-create happened. Plates uses created, edited and deleted; skipped and failed are always [] here, but they are always present, so one script reads all four elements the same way.

counts. success is created + edited + deleted — the number of plates successfully acted on. A Create with Override reports **2**, because it deleted one plate and created another. failed is the length of result.failed. warnings and errors count *messages*, not plates. Note that skipped has no count of its own, so success + failed does not necessarily cover every row — read result.skipped.length when you need it.

Reading steps. This matters most on Edit Plate. A normal edit is one save; a Description change cannot be saved in place, so the element renames the original aside, posts a replacement, and deletes the original — and steps shows exactly

that, in order. Each entry has step (stage number from 1), description, status (ok, warning or error), the method and endpoint of its first call, and items — **one entry per HTTP call the stage made**, each holding that call's own method, endpoint, status, request (the body sent) and response (Phoenix's reply).

So the batch size is items.length, and you can diff what was sent against what came back without the element having to echo either into a separate key. A stage that touches no HTTP at all — one that only rearranges data locally — reports items: [] and method: null.Three things to know: **a stage that did not run is absent**, not listed as skipped — so the presence of Delete existing (Override) is how you tell a replacement from a plain create; **a stage carries the warning it raised**, so when the cleanup delete fails and leaves an orphan under the temp name, it is the Delete original (DELETE) stage that reads "warning"; and **a call made before the first named stage** — the library lookup Override needs — lands in a stage named after the action, so 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.

5.2 <jobName>-error.json

Written when Phoenix reported an error (for example 400 or 404), or when the retries described below were exhausted.

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:


```
  {
    "action": "Edit Plate",
    "status": "error",
    "counts": { "success": 0, "warnings": 0, "errors": 1, "failed": 0 },
    "messages": {
      "warnings": [],
      "errors": [ { "step": 2, "item": 1, "message": "HTTP 404: Plate not found" } ]
    },
    "steps": [
      { "step": 1, "method": "GET", "endpoint": "/phoenix/libraries/plates",
        "status": "ok", "description": "Select plates", "items": [ "…1 call" ] },
      { "step": 2, "method": "GET", "endpoint": "/phoenix/libraries/plates/{plateid}",
        "status": "error", "description": "Fetch plate (GET)",
        "items": [
          { "item": 1, "method": "GET", "endpoint": "/phoenix/libraries/plates/{plateid}",
            "status": "error", "request": null,
            "response": {
              "success": false,
              "status-code": 404,
              "errors": [
                { "id": 404, "text": "Plate not found", "action": "GET /phoenix/libraries/plates/plate-
missing" }
              ],
              "warnings": [],
              "resources": []
            } }
        ] }
    ],
    "result": { "created": [], "edited": [], "deleted": [], "skipped": [], "failed": [] },
    "schema": { "app": "PhoenixPlates", "version": "1.0" }
  }
```


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

> All five result buckets are [] on this branch even when the action had already written some plates before it failed — read steps to see how far it got. This matters most on a Description change: if Create replacement (POST) succeeded but a later stage failed, a new plate exists even though result.created is empty.

Validation errors that come from the element itself — a missing required Name on Create, a blank Value on Delete — route to Error and write a clear message to the flow log, but do **not** attach an error JSON file.

5.3 503 retry behaviour

Phoenix reports “service at maximum concurrency” with HTTP 503 when it is overloaded. Phoenix Plates handles this automatically: it retries up to **5 times** with an increasing delay (roughly 10, 15, 20, 25 seconds). Only after the retries are exhausted does the job route to Error with the error JSON attached.

5.4 Concurrency safety

All four Phoenix library apps (Plates, Stocks, Dies, Things) coordinate through one shared on-disk lock so that only one write to the Phoenix library happens at a time — concurrent writes can corrupt Phoenix’s data. This is invisible in normal operation; you don’t configure anything. Read-only Find / Check for actions don’t take the lock and run fully concurrently. The lock recovers automatically if a job crashes mid-operation.

6. Troubleshooting & FAQ

My Check for Plate came out the Error port even though the flow ran fine.

That is how Check answers “no.” A Check for Plate that matches zero plates routes to Error on purpose, so you can branch on “this plate does not exist → go create it.” Use **Find Plate** instead if you want a match count without Error routing (Find returns `[]` and stays on Success).

Switch says my Delete Plate failed and I left Value blank.

That is by design. Delete refuses a blank Value so you can’t wipe the whole library by accident. Set a selector that matches only the plates you mean to remove.

My width / height didn’t take.

These are unit-bearing strings — send `23in` or `584mm`, not a bare number with the unit configured elsewhere. Phoenix reads the unit from the value itself.

I edited the Description and nothing changed.

Phoenix cannot write a description onto an existing plate — it accepts the request, answers success, and stores nothing. The element works around it by re-creating the plate, which gives it a **new id**. See 4.4.

My Create Plate failed with “An error occurred while trying to save Plate item with ID …”.

That is Phoenix’s message when a plate is missing something it requires. In practice it means Width or Height was blank; both are mandatory even though Phoenix’s own documentation only marks Name as required. Both arrive pre-filled with `1"`, so this only happens if you clear one. The element checks first and fails with a message naming the missing field, so if you see the raw Phoenix wording instead, it is coming from something else — check the Error output’s JSON for the full response.

My distortion came out as 0.998 instead of 99.8.

That is correct. You enter distortion as a percentage and Phoenix stores it as a multiplier, so `99.8%` is saved as `0.998` and shows as `99.8%` in Phoenix. Entering `1.0` is refused, because as a percentage it would mean a 100× shrink.

My distortion or price ended up as null / NaN.

Those fields are numeric. If a Switch variable resolves to something non-numeric, the job now fails with a message naming the field rather than sending an invalid body. Make sure the field resolves to a plain decimal — a percentage like `99.8` for distortion, or an amount like `12.50` for price.

Can I use Switch variables in the fields?

Yes — every text field accepts Switch variables and inline script expressions, including the selector Value, so the element can be fully data-driven from each job's metadata.

Can I run multiple Phoenix elements at once against the same server?

Yes. Phoenix Plates coordinates with the other Phoenix library apps on the same machine to avoid write conflicts (see 5.4). No configuration needed.