Phoenix Dies

User Manual

V1.0

Table of Contents

1. Overview

Phoenix Dies is an Enfocus Switch flow element that lets you read from and write to two Phoenix libraries — **Die Designs** and **Templates** — directly from a Switch flow. A *die design* is a cutting die imported from a CAD file; a *template* is a saved Phoenix layout imported from a template file. Anything you would otherwise do by hand in those libraries — importing a new die from a CAD file, checking whether one is already there, clearing out obsolete entries — you can do automatically as part of a job's journey through Switch.

A single Phoenix Dies element performs any one of **8 actions**: **Find**, **Check for**, **Create**, or **Delete**, each against either **Die Design** or **Template**. You pick the action from a dropdown; the rest of the form changes to show only the fields that action needs.

Creating an entry means **importing a file**. There is no way to type a die into existence — you point the element at a CAD or template file on disk and Phoenix reads the geometry out of it. That single fact shapes most of the form.

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 die design is or how templates are used in imposition, please refer to Phoenix's documentation.

2. Quick start

Drop a Phoenix Dies 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 `<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.

A minimal first flow: set **Action = Create Die Design**, put a CAD file path in **File path**, leave everything else alone, and run a job. The die appears in your Die Designs library under the file's own name.

3. Concepts you'll use in every action

3.1 Two libraries, one element

One Phoenix Dies element covers both libraries. The library is part of the action name, not a separate field — you choose **Create Die Design** or **Create Template**, not "Create" plus a target dropdown.

	Die Designs	**Templates**
What it holds	A cutting die imported from a CAD file	A saved Phoenix layout imported from a template file
Source file on Create	ARD, CFF2, DDES2, DDES3, DXF, MFG, PDF	A Phoenix template file
Extra Create fields	Die name (in CAD)	Import die designs, Die designs folder

Everything else — the selector, folders, Override, presets, routing — behaves identically for both.

Each entry, once imported, has these fields. You can filter on any of the first five:

Field	**Notes**
id	Server-assigned GUID. Read-only — you receive it back on Create and in Find results.
name	The entry's name in the library, **including any folder prefix** (see 3.4).
external-id	Optional identifier for cross-referencing with an MIS or ERP.
description	Free-text description.
path	The **source file on disk** the entry was imported from — not a location inside Phoenix.
width / height	Read from the imported file.

You cannot set external-id, description, width or height when creating. Phoenix derives them from the imported file. They appear in the selector because they exist on entries you read back, but there are deliberately no form fields for them on Create.

3.2 Why there is no Edit

The other Phoenix library apps have an Edit action. Phoenix Dies does not, because **Phoenix provides no way to modify a die design or template in place.** There is nothing for an Edit action to call.

To replace an entry, use **Create … with Override existing = Yes**. That deletes the same-named entry first, then imports the file fresh. See 3.6.

3.3 The selector pattern (By / Operator / Value)

To pick which entries an action operates on, the form uses three fields that always come together:

Field	What it does
By	The field to look at. One of: ID, Name, External ID, Description, Path.
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 entry whose name begins with "23x29".

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

- A **blank Value** with Equals matches only entries whose field is literally empty (usually none). A blank Value with Contains matches *every* entry — 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**, and **Delete**. **Create** has no selector — it always imports a new entry.

> ⚠️ **Path is the CAD file, not the Phoenix folder.** This is the single most common mix-up. See 3.4.

3.4 Folders live inside the Name

Phoenix organises both libraries into folders, but there is **no folder field**. A folder is a forward-slash prefix on the **Name**:

```
Cartons/23x29 Carton Die
```

That is the only way to put an entry into a folder, and the slash-prefixed form is what Phoenix stores. Three consequences:

Searching for a folder means searching on Name.

To find…	By	Operator	Value
everything in a folder	Name	Starts with	`Cartons/`
a folder anywhere in the library	Name	Contains	`/Cartons/`
one exact entry inside a folder	Name	Equals	`Cartons/23x29 Carton Die`
everything imported from one folder on disk	Path	Contains	`C:\hotfolder\`

An **exact** name search must include the folder. `Name Equals 23x29 Carton Die` will **not** find an entry stored as `Cartons/23x29 Carton Die`.

"Create folder" is a permission, not an instruction. It does not create anything by itself — it authorises Phoenix to create the folder you named in **Name**.

Name	Create folder	What happens
`Cartons/23x29 Carton Die`	Yes	Entry created inside a new `Cartons` folder
`Cartons/23x29 Carton Die`	No	**Job fails** — *Folder item 'Cartons' was not found in the library*
`23x29 Carton Die`	Yes	Entry created at the top level; no folder is created
`Cartons/Sleeves/23x29 Die`	Yes	Both `Cartons` and `Sleeves` created

A missing folder is a hard failure, not a fallback — Phoenix will not quietly put the entry at the top level instead.

Phoenix cannot delete folders. Folders can only be created. Deleting every entry in a folder leaves the (still usable) empty folder behind, and nothing in this element — or in any Switch flow — can remove it. Empty folders have to be deleted by hand in the Phoenix interface. This matters most when a flow builds folder names from a Switch variable: every distinct value that flow has ever produced leaves its own folder behind, so plan on an occasional tidy-up.

3.5 Bulk operations

Delete acts on **every** entry the selector matches, not just the first. If **By = Name, Operator = Starts with, Value = Cartons/** matches 12 entries, all 12 are deleted. This is by design, so you can clear a whole folder in one element. To act on exactly one entry, select **By = ID** with its GUID — IDs are unique.

3.6 The "Override existing" toggle

Both Create actions have an **Override existing** Yes/No field. When set to **Yes**, the element first deletes any existing entry with the same **Name**, then imports the new one — an idempotent "make sure this die exists, freshly imported from this file" operation. This is also the closest thing to an Edit (see 3.2).

Create Template has a **second** override, **Override existing die designs**, covered in 4.4.1. The two are deliberately separate: *Override existing* replaces the template entry, *Override existing die designs* clears the die designs a previous import of it put in the Die Designs library. They act on different libraries, so one is never a safe stand-in for the other.

Three things to know:

- **Override matches the whole Name, folder included.** `Cartons/My Die` only clears a same-named entry *inside* `Cartons/`. A bare `My Die` will not match it. That is usually what you want — override is scoped to the folder you are importing into — but it does mean the same leaf name can exist in several folders and Override touches only one.
- **An override that actually replaces something routes the job to Warning.** See 3.8. Nothing is wrong — it is how a flow tells "this added a new entry" apart from "this destroyed and rebuilt one". If nothing matched, there was nothing to replace and the job stays on Success.
- **Override deletes first, then imports.** If the import then fails, the old entry is already gone and the new one was never created — you end up with neither, and the job routes to Error. Most likely cause on Create Template is a die-design collision (4.4.1). Re-run the job once the cause is fixed.

When Override actually deletes something, the deleted entry appears in the result JSON as a row in `result.deleted`, carrying its id and name.

3.7 Presets

Preset is an optional Phoenix import preset to apply when reading the file. Leave it blank to use Phoenix's defaults.

Click **Select from Library** to fetch the live catalogue from your Phoenix server. What you get depends on the action:

- **Create Die Design** — presets from all six CAD formats, each shown as `Name (TYPE)`, e.g. `MyCadPreset (ARD)`. The suffix tells you which format the preset

belongs to; it is stripped before the request is sent, so Phoenix only ever sees the bare name.

- **Create Template** — PDF presets, listed with no suffix.

If the server is unreachable the picker is simply empty rather than erroring, and you can still type a preset name by hand or drive it from a Switch variable.

3.8 Connection routing summary

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

Connection	When
Success	The action completed as asked.
Warning	The action completed, but it **destroyed something** on the way. Both Create actions route here when an override actually deleted an existing entry: **Override existing** replacing a die design or template, or **Override existing die designs** clearing die designs from the folder. The reason is in `warnings[]` in the result JSON, one line per entry deleted.
Error	The action failed — an error from Phoenix, a missing required input (including a blank **Name** on either Create), a Delete with a blank Value, or a Delete whose selector matched nothing. A **Check for** that matches nothing also routes here; that is how Check answers "no".

Warning is about *replacement*, not about failure: the import succeeded and the library is in the state you asked for. It exists so a flow can treat "imported something new" and "overwrote what was there" differently — routing the second to an approval step or an audit log, for instance. An override that matched nothing does not warn, so a job that always warns is telling you it is replacing every time.

Draw the Warning connection. If you leave it undrawn, a job that overrides something has nowhere to go.

4. Actions

The Action dropdown at the top of the property panel chooses one of these 8 operations, grouped by verb:

> Find Die Design · Find Template
> Check for Die Design · Check for Template
> Create Die Design · Create Template
> Delete Die Design · Delete Template

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. The **Result JSON** block shows a realistic example of the file attached to the job.

4.1 Find Die Design / Find Template

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

Properties

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


```
[
  {
    "id": "550e8400-e29b-41d4-a716-446655440000",
    "name": "Cartons/23x29 Carton Die",
    "external-id": "DIE-001",
    "description": "RSC carton, B-flute",
    "path": "C:/dies/carton.ard",
    "width": "584mm",
    "height": "737mm"
  }
]
```


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

4.2 Check for Die Design / Check for Template

What it does. Tells you, with a Yes/No outcome, whether any entry matches. 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 die missing?" trivial to branch on.

Properties

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

What it does. Imports a CAD file into the Die Designs library. **Name** and **File path** are both required. If **Override existing = Yes** and an entry with the same Name already exists, it is deleted first.

> ⚠️ **Name is required, despite what you may expect.** Phoenix does *not* fall back to the source file name. Left blank, it stores an entry with no name at all — invisible to every Find, Override and Delete that searches by name, and impossible to clean up from a flow. Rather than let that happen, the job fails with `Create Die Design: Name is required`. Feed Name from a Switch variable (the job name, or a value pulled from your MIS) if you don't want to type one.

Properties

Field	Type	Required	Default	Notes
Name	Single-line	**Yes**	(blank)	Name for the new entry. Add a forward slash to place it in a folder, e.g. Cartons/23x29 Carton Die. Blank fails the job.
Override existing	Yes/No	No	No	If Yes, delete any existing entry with this exact Name first. The match includes the folder, so only an entry in the same folder is replaced.
File path	Single-line	**Yes**	(blank)	Full path to the source CAD file: ARD, CFF2, DDES2/3, DXF, MFG, PDF. Example: C:/dies/carton.ard.
Create folder	Yes/No	No	No	Permission for Phoenix to create the folder named in **Name**. Creates nothing on its own.
Preset	Single-line	No	(blank)	Optional Phoenix import preset. Use **Select from Library** to pick one.
Die name (in CAD)	Single-line	No	(blank)	Which die to import when the CAD file contains several. Blank takes the first. Example: DIE_A.

Result JSON


```
  {
    "action": "Create Die Design",
    "status": "ok",
    "counts": { "success": 1, "warnings": 0, "errors": 0, "failed": 0 },
    "messages": { "warnings": [], "errors": [] },
    "steps": [
     { "step": 1, "method": "POST", "endpoint": "/phoenix/libraries/diedesigns",
       "status": "ok", "description": "Import Die Design (POST)",
       "items": [
         { "item": 1, "method": "POST", "endpoint": "/phoenix/libraries/diedesigns",
           "status": "ok",
           "request": { "path": "C:/dies/carton.ard", "name": "Cartons/23x29 Carton Die", "create-
folder": true },
           "response": { "success": true, "resources": ["/phoenix/libraries/diedesigns/550e8400-
e29b-41d4-a716-446655440000"] } }
        ] }
    ],
    "result": {
      "created": [ { "id": "550e8400-e29b-41d4-a716-446655440000", "previousId": null, "name":
"Cartons/23x29 Carton Die" } ],
      "edited":  [],
      "deleted": [],
      "skipped": [],
      "failed":  []
    },
    "schema": { "app": "PhoenixDies", "version": "1.0" }
  }
```


- `result.created` — the imported entry, as `{ id, previousId, name }`. `previousId` is `null` on an import.
- `result.deleted` — what Override removed on the way in, `[]` when nothing was. **An Override delete**
  counts: `counts.success` becomes 2 when Override replaced an existing entry, because two entries were
  successfully acted on.
- `steps[0].items[0].request` is exactly what was sent to Phoenix — including the source path and the
  resolved folder — and `.response` is the reply, so you can diff the two without the element echoing
  either into a separate key.

4.4 Create Template

What it does. Imports a Phoenix template file into the Templates library. Same shape as Create Die Design, with two extra fields for the die designs contained in the template. **Name** and **File path** are both required.

Properties

Field	Type	Required	Default	Notes
Name	Single-line	**Yes**	(blank)	Name for the new entry; slash-prefix for a folder. Blank fails the job (see 4.3). Also the default **Die designs folder**.
Override existing	Yes/No	No	No	If Yes, delete any existing entry with this exact Name first. The match includes the folder, so only an entry in the same folder is replaced.
File path	Single-line	**Yes**	(blank)	Full path to the source Phoenix template file.
Create folder	Yes/No	No	No	Permission for Phoenix to create the folder named in **Name** — **and** one you type into Die designs folder. Creates nothing on its own.
Preset	Single-line	No	(blank)	Optional Phoenix import preset. Use **Select from Library** to pick one.
Import die designs	Yes/No	No	No	If Yes, die designs found inside the template are also imported into the Die Designs library. See 4.4.1 for how they are named.
Die designs folder	Single-line	No	**the template Name**	Folder in the Die Designs library for those die designs. Only shown when Import die designs is Yes. Leave blank to use the template's Name, created automatically.
Override existing die designs	Yes/No	No	No	If Yes, delete everything already in that folder before importing. Needed to re-import a template. Only shown when Import die designs is Yes.

4.4.1 How imported die designs are named

Turning **Import die designs** on hands the naming to Phoenix — it does not read a name out of your template file, it invents one. Two rules cover it, and both counters restart at 1 for **every** template you import:

The die…	is named
appears once in the template	S-1, S-2, S-3, …
appears more than once (multi-up)	1, 2, 3, …

A template with both kinds runs both sequences at once: one 2-up die plus two different one-off dies gives you 1, S-1 and S-2.

Three things that surprise people:

- **Repeats are merged.** The same shape three times on the sheet produces **one** die design, not three. So you will often get fewer die designs than there are shapes on the sheet.
- **Numbering follows position on the sheet, left to right** — not the order the shapes were drawn. The leftmost one-off die is S-1, the next to its right is S-2. That makes the names predictable enough to reference later in the flow.
- **The names are not unique across templates.** Every template starts again at S-1.

That last point is why **Die designs folder defaults to the template's Name**. Two templates importing into the same folder would both try to create S-1, and Phoenix rejects the second import outright:

```
Die Design item with name 'Cartons/S-1' already exists in the library
```

Nothing is renamed and nothing is skipped — the whole import fails. Defaulting the folder to the Name gives every template its own namespace, so a flow that imports templates in a loop just works. With Name RSC 4-up you get:

```
RSC 4-up/S-1
RSC 4-up/S-2
```

The folder is created for you, so you do **not** need to set *Create folder* for this. A folder in the Name comes along too: Name Cartons/RSC 4-up puts the die designs in Cartons/RSC 4-up/.

Fill Die designs folder in only when you want to override that — for example to collect several templates' dies in one place. A folder you type in yourself follows the normal rule below, and you take on the collision risk.

Re-importing a template you already imported

This needs **Override existing die designs = Yes**, and it is the one part of the form people miss.

Deleting a template does not delete the die designs it produced. They are separate library entries and they stay behind. So on the second import Phoenix hits

the leftover S-1 and rejects the whole thing — even with *Override existing* set to Yes, because that only replaces the template entry:

```
Die Design item with name 'RSC 4-up/S-1' already exists in the library
```

Setting **Override existing die designs = Yes** deletes everything in the die designs folder first, so the re-import gets a clean slate. To re-import a template repeatedly — a hot folder that keeps receiving updated versions of the same template, say — set **both** override toggles to Yes and the job becomes safely repeatable.

Override existing	Override existing die designs	Re-importing the same template
No	No	Fails: the template name already exists
Yes	No	Fails: '…/S-1' already exists
Yes	Yes	Works, every time. Routes to Warning

> ⚠️ **It deletes the whole folder's contents, not just this template's.** With the automatic per-template folder that is exactly this template's die designs and nothing else, which is what makes it safe. If you typed your own **Die designs folder** and share it between templates, turning this on **deletes the other templates' die designs too**. Either leave Die designs folder blank, or leave this toggle off and clear the folder yourself with **Delete Die Design**, *By* Name, *Operator* Starts with, *Value* <your folder>/.

Every die design deleted this way is a row in result.deleted in the result JSON — carrying its id and name — and sends the job down the **Warning** connection (see 3.8).

> ⚠️ **A folder you type needs Create folder too.** *Create folder* is one global permission covering both **Name** and a **Die designs folder** you supplied. Because it defaults to **No**, typing a new Die designs folder without switching Create folder to Yes fails the job — even when Name itself contains no folder at all. This does not apply to the automatic default, which brings its own permission.

> 💡 Phoenix cannot delete folders (see 3.4), so a flow that imports many templates will leave a folder behind for each one. They are harmless, but they accumulate — plan on an occasional tidy-up in the Phoenix interface.

Result JSON

Identical in shape to 4.3, with `app` and `action` naming the template import:


```
  {
    "action": "Create Template",
    "status": "ok",
    "counts": { "success": 1, "warnings": 0, "errors": 0, "failed": 0 },
    "messages": { "warnings": [], "errors": [] },
    "steps": [
      { "step": 1, "method": "POST", "endpoint": "/phoenix/libraries/templates",
        "status": "ok", "description": "Import Template (POST)",
        "items": [
          { "item": 1, "method": "POST", "endpoint": "/phoenix/libraries/templates",
            "status": "ok",
            "request": { "path": "C:/templates/rsc-4up.pdf", "name": "RSC 4-up", "die-designs-
folder": "RSC 4-up" },
            "response": { "success": true, "resources": ["/phoenix/libraries/templates/7c9e6679-
7425-40de-944b-e07fc1f90ae7"] } }
        ] }
    ],
    "result": {
      "created": [ { "id": "7c9e6679-7425-40de-944b-e07fc1f90ae7", "previousId": null, "name":
"RSC 4-up" } ],
      "edited":  [],
      "deleted": [],
      "skipped": [],
      "failed":  []
    },
    "schema": { "app": "PhoenixDies", "version": "1.0" }
  }
```


Both override passes report what they destroyed in `result.deleted`. On a re-import with both toggles on, the template clash and every stale die design land there as real rows — so you can count and name them rather than reading them out of prose. Each pass is also its own stage, and the warning it raised marks that stage `"warning"`:

```
  {
    "action": "Create Template",
    "status": "warning",
    "counts": { "success": 3, "warnings": 2, "errors": 0, "failed": 0 },
    "messages": {
      "warnings": [
        { "step": 2, "item": 1,
          "message": "Create Template — Override deleted the existing \"RSC 4-up\"
(id=0b38bd9a-…) before importing." },
        { "step": 3, "item": 1,
          "message": "Create Template — Override existing die designs deleted \"RSC 4-up/S-1\"
(id=2fbf2c1c-…) before importing." }
      ],
      "errors": []
    },
    "steps": [
      { "step": 1, "method": "GET", "endpoint": "/phoenix/libraries/templates",
        "status": "ok", "description": "Create Template", "items": [ "…2 lookup calls" ] },
      { "step": 2, "method": "DELETE", "endpoint": "/phoenix/libraries/templates/{templateid}",
        "status": "warning", "description": "Override existing Template", "items": [ "…1 call" ] },
      { "step": 3, "method": "DELETE", "endpoint": "/phoenix/libraries/diedesigns/{diedesignid}",
        "status": "warning", "description": "Override existing die designs", "items": [ "…1 call" ] },
      { "step": 4, "method": "POST", "endpoint": "/phoenix/libraries/templates",
        "status": "ok", "description": "Import Template (POST)", "items": [ "…1 call" ] }
    ],
    "result": {
      "created": [ { "id": "7c9e6679-…", "previousId": null, "name": "RSC 4-up" } ],
      "edited":  [],
      "deleted": [
        { "id": "0b38bd9a-…", "previousId": null, "name": "RSC 4-up" },
        { "id": "2fbf2c1c-…", "previousId": null, "name": "RSC 4-up/S-1" }
      ],
      "skipped": [],
      "failed":  []
    },
    "schema": { "app": "PhoenixDies", "version": "1.0" }
  }
```


(`items` is abbreviated above; each entry holds `{ item, method, endpoint, status, request, response }` as shown in full under 4.3.)

Note `counts.success` is **3** — one template created plus two entries deleted. Each override stage appears only when it actually deleted something, so you can tell *which* pass destroyed data, not just that something did. The two library lookups the overrides need run before any named stage, so they land in step 1 under the action's own name.

`steps[3].items[0].request` shows the folder that was actually used, so when **Die designs folder** was left blank you can read the defaulted value back out of the result. The flow log records it too:

Create Template — Die designs folder was blank; importing die designs into "RSC 4-up" so they cannot collide with another template's.

The die designs the import *created* are not listed — result.created describes the template. To get their ids, follow up with **Find Die Design**, *By* Name, *Operator* Starts with, *Value* RSC 4-up/.

4.5 Delete Die Design / Delete Template

What it does. Deletes every entry the selector matches. Two guards:

- **Value must not be blank.** A blank Value fails the job rather than emptying the library.
- **The selector must match something.** A Delete that matches nothing routes to **Error**, not Success — an empty delete reported as success is indistinguishable from one that actually removed something.

Remember that deleting every entry in a folder leaves the empty folder behind (see 3.4).

Properties

Field	Type	Required	Default	Notes
By	Dropdown	Yes	Name	ID, Name, External ID, Description, Path.
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 Die Design",
    "status": "ok",
    "counts": { "success": 2, "warnings": 0, "errors": 0, "failed": 0 },
    "messages": { "warnings": [], "errors": [] },
    "steps": [
      { "step": 1, "method": "GET", "endpoint": "/phoenix/libraries/diedesigns",
        "status": "ok", "description": "Select Die Designs", "items": [ "…1 call" ] },
      { "step": 2, "method": "DELETE", "endpoint": "/phoenix/libraries/diedesigns/{diedesignid}",
        "status": "ok", "description": "Delete Die Design", "items": [ "…2 calls, one per entry" ] }
    ],
    "result": {
      "created": [],
      "edited":  [],
      "deleted": [
        { "id": "550e8400-e29b-41d4-a716-446655440000", "previousId": null, "name":
"Cartons/23x29 Carton Die" },
        { "id": "6ba7b810-9dad-11d1-80b4-00c04fd430c8", "previousId": null, "name":
"Cartons/18x24 Carton Die" }
      ],
      "skipped": [],
      "failed":  []
    },
    "schema": { "app": "PhoenixDies", "version": "1.0" }
  }
```


(items is abbreviated above; each entry holds { item, method, endpoint, status, request, response } as shown in full under 4.3.)

Each deleted entry is one row in result.deleted and one item on the Delete Die Design stage, so the batch size is steps[1].items.length. Each item's response is null because Phoenix returns 204 No Content on delete.

5. Output files

Every job that passes through Phoenix Dies 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. A few rules across actions:

- For **Find**, the file is a JSON array at the top level ([] if nothing matched).
- For **Create** and **Delete**, the file is the shared envelope — the same shape used by Phoenix Stocks, Plates and Things, 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. Dies uses created (an import) and deleted (a Delete action, *and* whatever either Override pass cleared on the way into a Create); edited, 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 entries successfully acted on. A Create Template that overrode an existing template and one stale die design reports **3**, not 1, because three entries were acted on. failed is the length of result.failed. warnings and errors count *messages*, not entries. 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. Create Template can do three things — clear the existing template, clear the die designs a previous import left behind, then import — and steps reports whichever of those actually ran, 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 for `Override existing die designs`, `items.length` is the number of stale die designs cleared, and each item names the one it deleted. You can diff what was sent against what came back without the element echoing either into a separate key. A stage that touches no HTTP at all 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 `Override existing Template` is how you tell a replacement from a plain import; **a stage carries the warning it raised**, so when both override toggles fire you can see which pass destroyed what; and **calls made before the first named stage** — the library lookups the overrides need — land 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.

- For **Check for**, no file is written — the Yes/No outcome drives Success vs Error only.

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": "Create Die Design",
  "status": "error",
  "counts": { "success": 0, "warnings": 0, "errors": 1, "failed": 0 },
  "messages": {
    "warnings": [],
    "errors": [
      { "step": 1, "item": 1,
        "message": "HTTP 400: Folder item 'Cartons' was not found in the library" }
    ]
  },
  "steps": [
    { "step": 1, "method": "POST", "endpoint": "/phoenix/libraries/diedesigns",
      "status": "error", "description": "Import Die Design (POST)",
      "items": [
        { "item": 1, "method": "POST", "endpoint": "/phoenix/libraries/diedesigns",
          "status": "error",
          "request": { "path": "C:/dies/carton.ard", "name": "Cartons/23x29 Carton Die" },
          "response": {
            "success": false,
            "status-code": 400,
            "errors": [
              { "id": 400, "text": "Folder item 'Cartons' was not found in the library", "action": "POST
/phoenix/libraries/diedesigns" }
            ],
```

```
            "warnings": [],
            "resources": []
          } }
      ] }
  ],
  …
}
```

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, which on Dies is usually the answer — while messages.errors carries the readable version with the same step/item coordinates.

> All five result buckets are [] on this branch even when an override pass had already deleted something
> before the import failed — read steps to see which stages completed.

Validation errors that come from the element itself — a missing File path, 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 Dies 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 (Dies, Stocks, Plates, Things) coordinate through one shared on-disk lock so 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

I set "Create folder" to Yes but no folder appeared.

Create folder is a permission, not an instruction — it authorises Phoenix to create the folder you named in **Name**. If Name has no forward slash in it there is no folder to create, so nothing happens. Type the folder into Name: `Cartons/23x29 Carton Die`. See 3.4.

My job failed with "Folder item 'X' was not found in the library".

You named a folder in **Name** (or in **Die designs folder**) that doesn't exist yet, and **Create folder** is set to No. Set it to Yes. Phoenix will not quietly place the entry at the top level instead — a missing folder is a hard failure.

I filled in "Die designs folder" and the job failed, even though Name has no folder in it.

Create folder is one global permission covering both fields. Set it to Yes. This catches people out because Name looks folder-free, so the connection to *Create folder* isn't obvious.

Note this only applies to a folder **you type in**. Leave Die designs folder blank and the element uses the template's Name, creating that folder for you without needing *Create folder*. See 4.4.1.

My die designs came out called "S-1" and "S-2". Where do those names come from?

Phoenix invents them — nothing is read from your template file. A die used once becomes `S-1`, `S-2`, … numbered left to right across the sheet; a die repeated on the sheet becomes `1`, `2`, … and is imported once rather than once per repeat. See 4.4.1 for the full rules.

You cannot choose the names at import time. Either rename them afterwards in the Phoenix interface or import the die on its own with **Create Die Design**, where you set the Name yourself.

My second template import failed with "Die Design item with name 'X/S-1' already exists".

Two different causes, depending on whether it is the same template:

- **Re-importing the same template.** Its die designs from last time are still there — deleting a template does not remove them. Set **Override existing die designs = Yes**. See 4.4.1.
- **Two different templates sharing a folder.** The numbering restarts at `S-1` for every template, so they collide. This is why **Die designs folder** defaults to the template's Name; you only hit this if you typed a fixed folder in and reused it. Clear the field to get the automatic per-template folder back.

My job came out the Warning port after an import.

That is the intended signal that the import **replaced** something rather than only adding it — an override actually deleted an existing entry. Read `warnings[]` in the result JSON to see exactly what was deleted. If you don't need to distinguish the two cases, wire Warning to the same place as Success.

Override existing = Yes, but the second import still failed — and now the old template is gone too.

Override deletes first and imports second, so a failure in between leaves you with neither. On Create Template the usual cause is the die-design collision above; fix that (**Override existing die designs = Yes**) and re-run the job, which will import cleanly.

My Create job failed with "Name is required".

Phoenix has no usable blank-name behaviour — it does not fall back to the source file name. Left to itself it stores an entry with no name, which no Find, Override or Delete can ever match, so the element refuses the job instead. Fill in **Name**, or feed it from a Switch variable such as the job name.

Setting By = Path never matches my folder.

Path is the **source CAD file on disk** the entry was imported from, like `C:\dies\carton.dxf`. It has nothing to do with folders inside Phoenix. The Phoenix folder is part of the **Name** — use **By = Name, Operator = Starts with, Value = Cartons/**. See 3.4.

Name Equals my item name finds nothing, but I can see the item in Phoenix.

The stored name includes the folder prefix. An entry inside Cartons is stored as Cartons/23x29 Carton Die, so an exact search has to include the folder. Either search for the full name, or use **Contains** with the leaf name.

I deleted everything in a folder but the folder is still there.

Phoenix cannot delete folders — they can only be created. An empty folder has to be removed by hand in the Phoenix interface; nothing in this element or any Switch flow can do it. It's harmless, and if you later import into it you won't need *Create folder* again. See 3.4.

There's no Edit action. How do I change an existing die?

Phoenix provides no way to modify a die design or template in place, so there is nothing for an Edit action to call. Use **Create … with Override existing = Yes**, which deletes the same-named entry and re-imports the file. See 3.2.

What comes out of the Warning port?

Nothing, currently. Every situation this element can detect is either a clean success or a hard failure, so jobs come out Success or Error. The Warning port is present for consistency with the other Phoenix elements, which do use it — connect it if you like, but expect it to stay idle.

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

That's how Check answers "no". A Check that matches zero entries routes to Error on purpose, so you can branch on "this die doesn't exist → go import it". Use **Find** instead if you want a match count without Error routing — Find returns [] and stays on Success.

My Delete failed and I left Value blank.

By design. Delete refuses a blank Value so you can't empty the library by accident. A Delete whose selector matches nothing also fails, so a Delete reporting Success always means something was actually removed.

Override existing didn't replace my item.

Override matches the **whole** Name including the folder. Cartons/My Die won't match a top-level My Die, and vice versa. Check that the Name you're importing under is character-for-character the one already in the library.

Can I set the description or external ID when importing?

No. Phoenix derives description, external ID, width and height from the imported file, so there are no form fields for them. They appear in the selector because you can filter on them once the entry exists.

Can I use Switch variables in the fields?

Yes — every text field accepts Switch variables and inline script expressions, including the selector Value and File path, 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 Dies coordinates with the other Phoenix library apps on the same machine to avoid write conflicts (see 5.4). No configuration needed.