

ReSync API v1

Developer Guide

2026-10-28

generated from `docs/api/v1`

Contents

1. Introduction

2. Quickstart

3. A bid, end to end

4. Authentication and scopes

5. Sign in with ReSync

6. Endpoints

7. Errors

8. Events

9. Conventions

10. Changelog

1 Introduction

ReSync is a shared record of things and what has happened to them. This page explains the five ideas you need before the first call. Everything else in this pack builds on them.

The current version of this pack is published at docs.limnal.dev.

Resources

Everything in ReSync is a **resource**: a customer, a bid, a cabinet, a machine, a person, a company. A resource has an id, a name, an optional identifier of your own choosing, and a bag of **properties** — free-form key/value data.

Resource ids are integers, but the API returns them as **strings**. Some are larger than a JavaScript number can hold exactly. Keep them as strings and you will never have to think about it again.

Types

Every resource has one **type**, and a type is named by a short stable code — `cab_bid`, `cab_customer`, `cab_cat_wall`. The code is the thing you write in your own source; it never changes. Names and display labels do change, so do not key on them.

Types form a tree. Asking for a type in a list also returns resources of the types beneath it.

Ownership

Every resource is **owned** by one organization (or, where personal modeling is on, by a person). Ownership is not a permission setting you can adjust per request — it is a fact about the resource, and it is the boundary your key works inside.

The ledger

ReSync does not overwrite history. Every change appends an **event**: what happened, to which resource, when, who did it, and which system wrote it. Reading a resource shows you the current state; reading its events shows you how it got there, including the changes you made.

You do not write events directly. You call an ordinary endpoint and the event is recorded for you.

What your key is

Your key is a **credential owned by one organization**. That single fact decides what it can do:

- It is bound to that organization's resources. A resource owned by anyone else is not visible to it, by id or any other route. There is no "public" resource and no cross-organization read.

- It carries a **read set**, an **operate set** and a **find-or-create set**, each a list of type codes.
 - *Read set* — the types you may list and fetch.
 - *Operate set* — the types you may change. Operating implies reading, so a type in the operate set does not need to be repeated in the read set.
 - *Find-or-create set* — the types you may bring into existence, by your own identifier. Being able to change a type is not the same as being able to create one, so this list is separate.
- A type in none of the sets does not exist as far as your key is concerned.

You cannot change your own scopes. An administrator at the owning organization changes them, and every change is recorded on the ledger.

Call `GET /v1/api-keys/self` to see exactly what your key holds. That is the first call in the quickstart, and it is worth making it the first call your integration makes too.

When a **person** uses your app, they can sign in with their ReSync account and your app then acts as them, with their access and never more than your app is registered for: [sign-in.md](#).

Next: [quickstart.md](#) — the five calls every integration makes first, each with a real response. Then [walkthrough-bid.md](#) — one whole bid, from an empty start to the order it becomes, with a real response for every step.

2 Quickstart

The five calls every integration makes first, whatever it goes on to build: who your key is, which types it may touch, the vocabulary it may use, how to read a list, and how to record that something happened. Every response below is a real capture from a running server — nothing here is illustrative. The resources in the captures belong to a test organization; your ids, names and price levels will differ.

#	Door	Call
1	Self — who am I, and what may I touch?	GET /v1/api-keys/self
2	Types — map type codes to ids, read what a type declares	GET /v1/resource-types , GET /v1/resource-types/:id
3	Vocabulary — the kinds of connection and the events you may record	GET /v1/interaction-types , GET /v1/attestation-event-types
4	Read — a list, with each row's values	GET /v1/resources?typeKey=...&include=properties
5	Attest — record that something happened	POST /v1/resources/:id/attestations

Then [A bid, end to end](#) puts them to work: one whole bid, from an empty start to a converted order. [examples.http](#) has one runnable block per door. The latest copy of these pages lives at [docs.limnal.dev](#).

Set up

```
export BASE=https://api.limnal.dev
export TOKEN=... # the key your administrator gave you, shown once at creation
```

Every request carries the key as a bearer token, and nothing else:

```
curl -s -H "Authorization: Bearer $TOKEN" "$BASE/v1/..."
```

In code, one small wrapper is enough. This one never throws on an HTTP status — every answer, including a refusal, comes back as `{ status, json, header }`, so a step can decide what a `409` means for it — and it keeps the response headers, because the rate-limit headers are worth watching (see [Things to know](#)):

```

const BASE = process.env.RESYNC_BASE!           // https://api.limnal.dev
const TOKEN = process.env.RESYNC_TOKEN!         // from your secret store – never a command-line
argument, never logged

type Answer = { status: number; json: any; text: string; header: (name: string) => string | null }

async function api(method: string, path: string, body?: unknown): Promise<Answer> {
  let res: Response
  try {
    res = await fetch(`${BASE}${path}`, {
      method,
      headers: { authorization: `Bearer ${TOKEN}`, ...(body != null ? { 'content-type':
'application/json' } : {}) },
      body: body != null ? JSON.stringify(body) : undefined,
    })
  } catch (e) {
    return { status: 0, json: { error: 'network', message: (e as Error).message }, text: '',
header: () => null }
  }
  const text = await res.text()
  let json: any = null
  try { json = text ? JSON.parse(text) : null } catch { json = text }
  return { status: res.status, json, text, header: (name) => res.headers.get(name) }
}

```

1. Who am I, and what may I touch?

```
curl -s -H "Authorization: Bearer $TOKEN" "$BASE/v1/api-keys/self"
```

From `self.json` — status 200:

```

{
  "apiKeyId": 783,
  "name": "bid-app-docs",
  "scopes": {
    "read_types": [
      "cab_pricing_level",
      "cab_payment_terms",
      "cab_order",
      "cab_catalog_product",
      "cab_cat_accessory",
      "cab_cat_base",
      "cab_cat_molding",
      "cab_cat_office",
      "cab_cat_utility",
      "cab_cat_vanity",
      "cab_cat_wall",
      "cab_fam_base_sink",
      "cab_fam_bookcase",
      "cab_fam_crown_molding",
      "cab_fam_filler",
      "cab_fam_lazy_susan",
      "cab_fam_oven_cabinet",
      "cab_fam_panel",
      "cab_fam_post",
      "cab_fam_utility_cabinet",
      "cab_fam_vanity_combo",
      "cab_fam_wall_cabinet",
      "cab_fam_wall_drawer",
      "cab_fam_wall_pantry_pullout_filler",
      "cab_fam_wall_stacked",
      "door_style_lower",
      "door_style_upper",
      "drawer",
      "finish",
      "front",
      "hinge",
      "interior",
      "sheen",
      "slide",
      "species"
    ],
    "operate_types": [
      "cab_customer",
      "cab_bid",
      "cab_design_set",
      "cab_line",
      "cab_install",
      "cab_adjustment",
      "cab_modification"
    ],
    "find_or_create_types": [
      "cab_customer",
      "cab_bid",
      "cab_design_set",
      "cab_line",
      "cab_install",
      "cab_adjustment",
      "cab_modification"
    ],
    "created_owner_resource_id": 5
  },
  "actsAt": null,
  "owner": {
    "resourceId": 5,
    "name": "Popular Design Mill & Cabinet, Inc."
  }
}

```

```
}  
}
```

What to notice. `owner` is the organization your key belongs to — every resource you can reach is owned by it. `operate_types` is what you may change; `read_types` is what you may additionally read; `find_or_create_types` is what you may bring into existence. This key may create and change customers, bids, design sets and the four kinds of bid line, and only read the price book, the price levels, the payment terms and the orders a converted bid produces. A type code in none of the lists returns `403` everywhere, and that is working as intended.

Make this call at startup and log the result. When something later returns `403`, this is the answer.

2. Map type codes to ids

Doors take a type's `code` (`typeKey=cab_bid`); a few answers carry its numeric id. `GET /v1/resource-types` lists every type your key can read, each with its `code` and `resourceTypeId` — read it once and keep the code → id map. The full list is long (`resource-types-list.json` in the fixture set); `lifecycleId` narrows it to the types that move through one lifecycle — here, the bid's:

```
curl -s -H "Authorization: Bearer $TOKEN" "$BASE/v1/resource-types?lifecycleId=<bidLifecycleId>"
```

From `resource-types-list-by-lifecycle.json` — status 200:

```
[  
  {  
    "resourceTypeId": "176",  
    "code": "cab_bid",  
    "name": "Bid",  
    "description": null,  
    "propertySchema": {  
      "tax_rate_bp": {  
        "type": "number",  
        "label": "Tax rate (basis points)",  
        "derivable": false  
      }  
    },  
    "parameterDefinitions": {},  
    "labelTemplate": "{name}",  
    "isSystem": false,  
    "canBeAllocated": false,  
    "isConsumable": false,  
    "parentResourceTypeId": "170",  
    "parentName": "Shop Document",  
    "ownerResourceId": 5,  
    "lifecycleId": "108",  
    "orgId": "5",  
    "orgName": "Popular Design Mill & Cabinet, Inc."  
  }  
]
```

`propertySchema` is what the type declares about its values (`derivable: false` marks money ReSync never computes); `lifecycleId` is the lifecycle its statuses come from. One type in full also says what it allows as a member — here, a bid line can sit inside at most one design set:

```
curl -s -H "Authorization: Bearer $TOKEN" "$BASE/v1/resource-types/<lineTypeId>"
```

From `resource-type-by-id.json` — status 200:

```

{
  "resourceTypeId": "3425",
  "code": "cab_line",
  "name": "Bid Line",
  "description": null,
  "propertySchema": {
    "qty": {
      "type": "number",
      "label": "Quantity"
    },
    "area": {
      "type": "text",
      "label": "Area"
    },
    "taxable": {
      "type": "select",
      "label": "Taxable",
      "options": [
        "yes",
        "no"
      ]
    },
    "ext_cents": {
      "type": "number",
      "label": "Extended price (cents)",
      "derivable": false
    },
    "shop_note": {
      "type": "text",
      "label": "Shop note"
    },
    "unit_cents": {
      "type": "number",
      "label": "Unit price (cents)",
      "derivable": false
    },
    "customer_note": {
      "type": "text",
      "label": "Customer note"
    },
    "finish_interior": {
      "type": "select",
      "label": "Finished interior",
      "options": [
        "yes",
        "no"
      ]
    },
    "list_overridden": {
      "type": "select",
      "label": "List overridden",
      "options": [
        "yes",
        "no"
      ]
    },
    "override_list_cents": {
      "type": "number",
      "label": "Override list price (cents)",
      "derivable": false
    }
  },
  "nameTemplate": null,
  "parameterDefinitions": {},
  "expressionContext": {},
  "labelTemplate": "{name}",

```

```
"isSystem": false,
"canBeAllocated": false,
"isConsumable": false,
"defaultIsSerialized": false,
"defaultIsLotTracked": false,
"parentResourceTypeId": "170",
"defaultUnitTypeId": null,
"ownerResourceId": "5",
"lifecycleId": "656",
"orgName": "Popular Design Mill & Cabinet, Inc.",
"ownerOrgId": "1",
"canvasId": null,
"containmentCaps": [
  {
    "containmentCapId": 1,
    "parentTypeId": 1171,
    "maxParents": 1
  }
]
}
```

3. Read the vocabulary

Two lists name the words the rest of the API speaks. **Interaction types** are the kinds of connection between two resources — a bid's customer is a `bid_for` connection, its price level `priced_at` :

```
curl -s -H "Authorization: Bearer $TOKEN" "$BASE/v1/interaction-types"
```

From `interaction-types-read.json` — status 200:

```

{
  "interactionTypes": [
    {
      "interactionTypeId": 1,
      "name": "allocates",
      "affects": "availability",
      "isBidirectional": false,
      "isSystem": true,
      "affectsInventory": false,
      "affectsAvailability": true,
      "affectsScheduling": false,
      "connectable": true
    },
    {
      "interactionTypeId": 6,
      "name": "assigned_to",
      "affects": "availability",
      "isBidirectional": false,
      "isSystem": true,
      "affectsInventory": false,
      "affectsAvailability": true,
      "affectsScheduling": false,
      "connectable": true
    },
    {
      "interactionTypeId": 13,
      "name": "bid_for",
      "affects": "none",
      "isBidirectional": false,
      "isSystem": false,
      "affectsInventory": false,
      "affectsAvailability": false,
      "affectsScheduling": false,
      "connectable": true
    },
    {
      "interactionTypeId": 7,
      "name": "certified_in",
      "affects": "none",
      "isBidirectional": false,
      "isSystem": true,
      "affectsInventory": false,
      "affectsAvailability": false,
      "affectsScheduling": false,
      "connectable": true
    },
    {
      "interactionTypeId": 9,
      "name": "compatible_with",
      "affects": "none",
      "isBidirectional": true,
      "isSystem": true,
      "affectsInventory": false,
      "affectsAvailability": false,
      "affectsScheduling": false,
      "connectable": false
    },
    {
      "interactionTypeId": 2,
      "name": "consumes",
      "affects": "inventory",
      "isBidirectional": false,
      "isSystem": true,
      "affectsInventory": true,
      "affectsAvailability": false,
      "affectsScheduling": false,
    }
  ]
}

```

```

    "connectable": true
  },
  {
    "interactionTypeId": 5,
    "name": "depends_on",
    "affects": "scheduling",
    "isBidirectional": false,
    "isSystem": true,
    "affectsInventory": false,
    "affectsAvailability": false,
    "affectsScheduling": true,
    "connectable": true
  },
  {
    "interactionTypeId": 11,
    "name": "derives_from",
    "affects": "none",
    "isBidirectional": false,
    "isSystem": true,
    "affectsInventory": false,
    "affectsAvailability": false,
    "affectsScheduling": false,
    "connectable": true
  },
  {
    "interactionTypeId": 12,
    "name": "feeds",
    "affects": "none",
    "isBidirectional": false,
    "isSystem": true,
    "affectsInventory": false,
    "affectsAvailability": false,
    "affectsScheduling": false,
    "connectable": true
  },
  {
    "interactionTypeId": 10,
    "name": "incompatible_with",
    "affects": "none",
    "isBidirectional": true,
    "isSystem": true,
    "affectsInventory": false,
    "affectsAvailability": false,
    "affectsScheduling": false,
    "connectable": false
  },
  {
    "interactionTypeId": 8,
    "name": "licensed_for",
    "affects": "none",
    "isBidirectional": false,
    "isSystem": true,
    "affectsInventory": false,
    "affectsAvailability": false,
    "affectsScheduling": false,
    "connectable": true
  },
  {
    "interactionTypeId": 15,
    "name": "pays_on",
    "affects": "none",
    "isBidirectional": false,
    "isSystem": false,
    "affectsInventory": false,
    "affectsAvailability": false,
    "affectsScheduling": false,

```

```

    "connectable": true
  },
  {
    "interactionTypeId": 14,
    "name": "priced_at",
    "affects": "none",
    "isBidirectional": false,
    "isSystem": false,
    "affectsInventory": false,
    "affectsAvailability": false,
    "affectsScheduling": false,
    "connectable": true
  },
  {
    "interactionTypeId": 4,
    "name": "produces",
    "affects": "inventory",
    "isBidirectional": false,
    "isSystem": true,
    "affectsInventory": true,
    "affectsAvailability": false,
    "affectsScheduling": false,
    "connectable": true
  },
  {
    "interactionTypeId": 3,
    "name": "reserves",
    "affects": "availability",
    "isBidirectional": false,
    "isSystem": true,
    "affectsInventory": false,
    "affectsAvailability": true,
    "affectsScheduling": false,
    "connectable": true
  },
  {
    "interactionTypeId": 17,
    "name": "sells",
    "affects": "none",
    "isBidirectional": false,
    "isSystem": false,
    "affectsInventory": false,
    "affectsAvailability": false,
    "affectsScheduling": false,
    "connectable": true
  },
  {
    "interactionTypeId": 16,
    "name": "uses",
    "affects": "none",
    "isBidirectional": false,
    "isSystem": false,
    "affectsInventory": false,
    "affectsAvailability": false,
    "affectsScheduling": false,
    "connectable": true
  }
]
}

```

What to notice. `connectable: false` marks a kind you cannot draw yourself — `compatible_with` and `incompatible_with` are compatibility markings, kept by your administrator. A field already knows its own kind (its `defaultInteractionTypeId`, see [the walkthrough](#)); this list is for naming what you read back.

Attestation event types are the things you may record as having happened — a bid was sent, accepted, converted — each with the payload fields it declares:

```
curl -s -H "Authorization: Bearer $TOKEN" "$BASE/v1/attestation-event-types"
```

The full answer is `attestation-event-types-read.json` in the fixture set — one row per event type your key can see. Read it rather than caching it forever: a declaration can change, and the [changelog](#) says when.

4. Read a list

A list always needs a type. Add `include=properties` to get each row's values in the same request:

```
curl -s -H "Authorization: Bearer $TOKEN" "$BASE/v1/resources?
typeKey=cab_pricing_level&limit=5&include=properties"
```

From `list-include-properties.json` — status 200:

```

{
  "resources": [
    {
      "resourceId": "13806",
      "name": "Dealer",
      "identifier": null,
      "ownerResourceId": 5,
      "resourceTypeId": "1264",
      "resourceTypeName": "Price Level",
      "isSystem": false,
      "createdAt": "2026-09-13T08:53:20.358Z",
      "statusId": null,
      "statusName": null,
      "statusColor": null,
      "archivedAt": null,
      "icon": "box",
      "properties": {
        "name": "Dealer",
        "pct_bp": 4536
      },
      "confirmedKeys": {}
    },
    {
      "resourceId": "13807",
      "name": "High-Volume Contractor",
      "identifier": null,
      "ownerResourceId": 5,
      "resourceTypeId": "1264",
      "resourceTypeName": "Price Level",
      "isSystem": false,
      "createdAt": "2026-09-13T08:53:22.255Z",
      "statusId": null,
      "statusName": null,
      "statusColor": null,
      "archivedAt": null,
      "icon": "box",
      "properties": {
        "name": "High-Volume Contractor",
        "pct_bp": 5025
      },
      "confirmedKeys": {}
    },
    {
      "resourceId": "13808",
      "name": "Contractor",
      "identifier": null,
      "ownerResourceId": 5,
      "resourceTypeId": "1264",
      "resourceTypeName": "Price Level",
      "isSystem": false,
      "createdAt": "2026-09-13T08:53:24.139Z",
      "statusId": null,
      "statusName": null,
      "statusColor": null,
      "archivedAt": null,
      "icon": "box",
      "properties": {
        "name": "Contractor",
        "pct_bp": 5678
      },
      "confirmedKeys": {}
    },
    {
      "resourceId": "13809",
      "name": "Homeowner",
      "identifier": null,

```

```

    "ownerResourceId": 5,
    "resourceTypeId": "1264",
    "resourceTypeName": "Price Level",
    "isSystem": false,
    "createdAt": "2026-09-13T08:53:26.009Z",
    "statusId": null,
    "statusName": null,
    "statusColor": null,
    "archivedAt": null,
    "icon": "box",
    "properties": {
      "name": "Homeowner",
      "pct_bp": 6937
    },
    "confirmedKeys": {}
  },
  {
    "resourceId": "13810",
    "name": "Internal",
    "identifier": null,
    "ownerResourceId": 5,
    "resourceTypeId": "1264",
    "resourceTypeName": "Price Level",
    "isSystem": false,
    "createdAt": "2026-09-13T08:53:27.881Z",
    "statusId": null,
    "statusName": null,
    "statusColor": null,
    "archivedAt": null,
    "icon": "box",
    "properties": {
      "name": "Internal",
      "pct_bp": 0
    },
    "confirmedKeys": {}
  }
],
"nextCursor": "13810"
}

```

What to notice. `resourceId` is a **string**. `nextCursor` is an opaque token: pass it back as `&cursor=...` for the next page, and stop when it comes back `null`. Payment terms are the same call with `typeKey=cab_payment_terms`. ReSync stores these numbers; it never multiplies them — computing a price is your application's job.

5. Record what happened

An **attestation** records that something happened — here, that a bid was sent, with the totals your application computed — without changing the resource's own fields:

```

curl -s -X POST -H "Authorization: Bearer $TOKEN" -H 'Content-Type: application/json' \
  -d '{"eventType":"bid.sent","payload":
{"subtotal_cents":155027,"discount_cents":0,"delivery_cents":0,"tax_cents":12790,"install_cents":50
000,"total_cents":167817,"tax_rate_bp":825}}' \
  "$BASE/v1/resources/$BID/attestations"

```

From `bid-attest-sent.json` — status 201:

```
{
  "eventUlid": "01M5FWMHJY4Q8TB2KSHCS8Z9S9",
  "eventType": "bid.sent",
  "resourceId": 24911,
  "actorResourceId": 24907
}
```

What to notice. The answer names the event (`eventUlid`) and who it is attributed to: your key's own service identity, unless you name a person with `actorResourceId` . An event type that is not in your vocabulary, a payload that misses a declared field, or a resource whose status does not allow the event is refused — see [errors.md](#). Some events are recorded once only; sending one again is `409` .

Next: A bid, end to end

A bid, end to end — one customer, one bid, a design set with five lines, from Draft through Frozen, Sent, Accepted and Converted to the order ReSync creates. Every step quotes its capture.

Where to go next

- [walkthrough-bid.md](#) — a whole bid, door by door, and the things to know before you build one.
- [auth-and-scopes.md](#) — scopes, rate limits, request ids.
- [endpoints.md](#) — every door, with parameters and errors.
- [errors.md](#) — every error code and what to do about it.
- [conventions.md](#) — the rules that apply everywhere, and what you cannot do yet.

3 A bid, end to end

One whole bid on an integration key, from an empty start to the order ReSync creates when the bid converts: one customer, one bid, a design set with five lines, an installation line and a modification. Every response below is a real capture from one continuous run against a running server, in the order it happened — nothing here is illustrative. The resources are a test customer, "Harbor Homes", and its kitchen bid; your ids, names and price levels will differ.

Start with the [Quickstart](#): it sets up the token and the `api()` wrapper, and shows the five calls this page builds on — who your key is, the type codes, the vocabulary, a list read and an attestation.

#	Step	Status of the bid	Doors
1	Bring in the customer and the bid by your own identifiers	Draft	POST <code>/v1/resources/find-or-create</code> , PATCH <code>/v1/resources/:id</code>
2	Fill the bid's three fields	Draft	GET <code>.../ports</code> , GET <code>.../ports/:portId/legal-targets</code> , POST <code>/v1/resource-interactions</code>
3	A design set inside the bid, and its ten fields	Draft	POST <code>/v1/resources/:id/members</code> , legal-targets
4	Lines, each with its product	Draft	find-or-create, members, PATCH , legal-targets with <code>q</code>
5	Change or remove a field; the one-set limit	Draft	POST <code>/v1/resource-interactions</code> with <code>replaceInteractionId</code> , DELETE <code>/v1/resource-interactions/:id</code>
6	Installation and modification lines	Draft	find-or-create, members
7	Freeze bottom-up	Draft → lines and set Frozen	GET <code>/v1/state-transitions</code> , GET <code>.../requirements</code> , PATCH <code>.../status</code>
8	Submit	Sent	attestations, GET <code>.../requirements</code> , PATCH <code>.../status</code>
9	Accept	Accepted	attestations, GET <code>.../events</code>
10	Convert — one call; the move records the conversion	Converted	GET <code>.../requirements</code> , GET <code>.../advancement</code> , PATCH <code>.../status</code>
11	The order	Order: Received	GET <code>/v1/resources/:id</code> , <code>.../events</code> , <code>.../members</code> , GET <code>/v1/resources?typeKey=cab_order&q=...</code>
12	Accepted internally, by a named person	Accepted	attestations with <code>actorResourceId</code>

Steps that use the same doors again (the second field, the next line) are shown once.

1. Bring in the customer and the bid

`find-or-create` is the idempotent way in: the same identifier from your system always answers the same resource. Its body is **snake_case**, unlike everywhere else. The customer comes first (`type_key: "cab_customer"`), then the bid:

```
curl -s -X POST -H "Authorization: Bearer $TOKEN" -H 'Content-Type: application/json' \
-d '{"type_key": "cab_bid", "external_system": "pdmac-sql", "external_key": "docs-bid-1", "name": "Harbor Homes - kitchen"}' \
"$BASE/v1/resources/find-or-create"
```

From `bid-foc-create.json` — status 201:

```
{
  "resource_id": 24911,
  "created": true
}
```

Called again with the same identifier, it answers 200 with `"created": false` and changes nothing. `external_system` names an external system ReSync already knows; your administrator tells you which.

Set the bid's own values with a PATCH. `properties` **merges** — keys you do not send are left alone:

```
curl -s -X PATCH -H "Authorization: Bearer $TOKEN" -H 'Content-Type: application/json' \
-d '{"properties":{"tax_rate_bp":825}}' "$BASE/v1/resources/<bidId>"
```

From `bid-patch-tax.json` — status 200:

```
{
  "resourceId": "24911",
  "resourceTypeId": "176",
  "ownerResourceId": "5",
  "name": "Harbor Homes - kitchen",
  "identifier": null,
  "properties": {
    "tax_rate_bp": 825
  },
  "statusId": "280",
  "isVerified": false,
  "confirmedKeys": {},
  "createdAt": "2026-10-21T18:47:21.014Z"
}
```

Behind this call ReSync recorded an event saying which property changed, from what, to what, and that **your key** wrote it — see [events.md](#).

2. Fill the bid's three fields

A bid's customer, price level and payment terms are **fields**: a port that holds one choice. Three calls fill one:

1. GET `/v1/resources/$BID/ports` — find the port by its `portName` (`customer` , `level` , `terms`); keep its `resourcePortId` and `defaultInteractionTypeId` .
2. GET `/v1/resources/$BID/ports/$PORT/legal-targets?scope=type` — the resources the field may hold, each with a `verdict` and the target port to use (`viableTargetPorts[0]`). Narrow a long list with `q` and `limit` .
3. POST `/v1/resource-interactions` — the attachment.

```
curl -s -X POST -H "Authorization: Bearer $TOKEN" -H 'Content-Type: application/json' \
-d '{"sourceResourceId":<bidId>,"targetResourceId":<customerId>,"interactionTypeId":
<defaultInteractionTypeId>,"sourcePortId":<customerPortId>,"targetPortId":<targetPortId>}' \
"$BASE/v1/resource-interactions"
```

From `bid-field-customer.json` — status 201:

```
{
  "interactionId": 2720,
  "sourceResourceId": 24911,
  "targetResourceId": 24910,
  "interactionTypeId": 23,
  "contextId": null
}
```

The customer's own price level and payment terms are filled the same way, before the bid. Read the bid back — its `edges` list the three fields:

From `bid-get-draft.json` — status 200:

```

{
  "resourceId": "24912",
  "resourceTypeId": "176",
  "ownerResourceId": "5",
  "name": "Harbor Homes – kitchen",
  "identifier": null,
  "properties": {
    "tax_rate_bp": 825
  },
  "statusId": "280",
  "isVerified": false,
  "createdAt": "2026-10-24T00:14:12.321Z",
  "archivedAt": null,
  "derivedFromTypeId": null,
  "resourceTypeName": "Bid",
  "propertySchema": {
    "tax_rate_bp": {
      "type": "number",
      "label": "Tax rate (basis points)",
      "derivable": false
    }
  },
  "declaredPropertySchema": {
    "tax_rate_bp": {
      "type": "number",
      "label": "Tax rate (basis points)",
      "derivable": false
    }
  },
  "orgName": "Popular Design Mill & Cabinet, Inc.",
  "ownerOrgId": "1",
  "canvasMode": null,
  "statusName": "Draft",
  "statusColorCode": "#64748b",
  "icon": "box",
  "lifecycleId": 108,
  "canvasId": null,
  "derivedFrom": [],
  "derivatives": [],
  "derivedKeys": [],
  "edges": [
    {
      "interactionId": 2724,
      "typeName": "bid_for",
      "direction": "out",
      "otherResourceId": 24911,
      "otherResourceName": "Harbor Homes",
      "thisResourcePortId": 7714
    },
    {
      "interactionId": 2725,
      "typeName": "priced_at",
      "direction": "out",
      "otherResourceId": 13808,
      "otherResourceName": "Contractor",
      "thisResourcePortId": 7715
    },
    {
      "interactionId": 2726,
      "typeName": "pays_on",
      "direction": "out",
      "otherResourceId": 14013,
      "otherResourceName": "Net 30",
      "thisResourcePortId": 7716
    }
  ],
}

```

```
"otherEdges": [],
"confirmedKeys": [],
"readOnly": null
}
```

What to notice. `statusName` is `Draft`, and `readOnly` is `null`: everything about a Draft bid can still change.

3. A design set inside the bid

Create the set with `find-or-create` (`type_key`: `"cab_design_set"`), then put it inside the bid:

```
curl -s -X POST -H "Authorization: Bearer $TOKEN" -H 'Content-Type: application/json' \
-d '{"childResourceId":<setId>}' "$BASE/v1/resources/<bidId>/members"
```

From `members-enrol.json` — status 201:

```
{
  "componentId": 262526,
  "parentResourceId": 24911,
  "childResourceId": 24912,
  "componentRole": null,
  "sourceSlotId": null,
  "sourceRuleId": null,
  "componentStatus": "active",
  "statusChangedAt": "2026-10-21T18:47:22.424Z",
  "sortOrder": 0,
  "properties": {},
  "createdAt": "2026-10-21T18:47:22.424Z"
}
```

`childResourceId` is a JSON **number**. A design set has ten material and hardware fields — Species, Finish, Sheen, Interior, Upper Door Style, Lower Door Style, Drawer, Slide, Front, Hinge — filled like any field (step 2). Each list says, per candidate, whether it fits what the set already holds:

From `set-axis-legal-targets.json` — status 200:

```

{
  "sourcePort": {
    "portId": 7717,
    "portName": "Species",
    "direction": "out",
    "maxConnections": 1,
    "held": 0,
    "capRemaining": 1
  },
  "candidates": [
    {
      "resourceId": 7272,
      "name": "Outsource Specie",
      "resourceTypeId": 1404,
      "verdict": "legal",
      "reasons": [],
      "viableTargetPorts": [
        {
          "portId": 1501,
          "portName": "DesignSets",
          "direction": "in"
        }
      ]
    },
    {
      "resourceId": 7271,
      "name": "TBD",
      "resourceTypeId": 1404,
      "verdict": "legal",
      "reasons": [],
      "viableTargetPorts": [
        {
          "portId": 1500,
          "portName": "DesignSets",
          "direction": "in"
        }
      ]
    },
    {
      "resourceId": 7270,
      "name": "Rustic Walnut",
      "resourceTypeId": 1404,
      "verdict": "legal",
      "reasons": [],
      "viableTargetPorts": [
        {
          "portId": 1499,
          "portName": "DesignSets",
          "direction": "in"
        }
      ]
    },
    {
      "resourceId": 7269,
      "name": "HDF/Paint Grade",
      "resourceTypeId": 1404,
      "verdict": "legal",
      "reasons": [],
      "viableTargetPorts": [
        {
          "portId": 1498,
          "portName": "DesignSets",
          "direction": "in"
        }
      ]
    }
  ],
}

```

```
{
  "resourceId": 7268,
  "name": "Rift White Oak",
  "resourceTypeId": 1404,
  "verdict": "legal",
  "reasons": [],
  "viableTargetPorts": [
    {
      "portId": 1497,
      "portName": "DesignSets",
      "direction": "in"
    }
  ]
},
"truncated": true,
"total": 17
}
```

A `verdict` other than `legal` comes with `reasons` naming why — a value the set's other choices rule out is `compat_deny`. Legal-targets is your compatibility check: it gives the same answer the attachment would, before you make it. There is no bulk read of compatibility rules for a key; ask the field.

4. Lines, each with its product

Look the product up in the price book first — a list over its type with `q`:

```
curl -s -H "Authorization: Bearer $TOKEN" "$BASE/v1/resources?
typeKey=cab_catalog_product&q=AP0&limit=5&include=properties"
```

The answer is `list-products-q.json` in the fixture set: each product's `identifier` and its `list_cents`. Then, for each line: `find-or-create` (`type_key`: `"cab_line"`), `POST /v1/resources/$SET/members`, a `PATCH` for its quantities and prices, and its `product` field.

From `line-foc-create.json` — status 201:

```
{
  "resource_id": 24913,
  "created": true
}
```

The `product` field's list, narrowed to the product's identifier:

```
curl -s -H "Authorization: Bearer $TOKEN" "$BASE/v1/resources/<lineId>/ports/<productPortId>/legal-
targets?scope=type&q=AP006&limit=5"
```

From `legal-targets-q.json` — status 200:

```
{
  "sourcePort": {
    "portId": 7736,
    "portName": "product",
    "direction": "out",
    "maxConnections": 1,
    "held": 0,
    "capRemaining": 1
  },
  "candidates": [
    {
      "resourceId": 9009,
      "name": "AP006",
      "resourceTypeId": 182,
      "verdict": "legal",
      "reasons": [],
      "viableTargetPorts": [
        {
          "portId": 4606,
          "portName": "Sold as",
          "direction": "in"
        }
      ]
    }
  ],
  "truncated": false,
  "total": 1
}
```

From `line-product-field.json` — status 201:

```
{
  "interactionId": 2733,
  "sourceResourceId": 24913,
  "targetResourceId": 9009,
  "interactionTypeId": 278,
  "contextId": null
}
```

A line never carries the product identifier itself; it points at the product. A line inherits each of the set's ten fields unless it chooses its own — its own list for one of them is `line-axis-legal-targets.json` in the fixture set.

Save the whole bid in one request

Sections 1 to 4 and 6 build the bid one resource at a time. The same `find-or-create` door can take the whole bid at once: each resource carries its `fields` — by `portName`, pointing at the resource you chose — and the resources inside it as `children`. The ids are the ones you would have picked in §2–§4: the customer, the price level and terms, the ten design choices (each from its field's legal targets) and each line's product. This is the body of the capture below, with lines three to five left out here — they have the same shape as line two:

```

{
  "type_key": "cab_bid",
  "external_system": "pdmac-sql",
  "external_key": "docs-tree-bid",
  "name": "Harbor Homes – laundry",
  "properties": {
    "tax_rate_bp": 825
  },
  "fields": {
    "customer": 24910,
    "level": 13808,
    "terms": 14013
  },
  "children": [
    {
      "type_key": "cab_design_set",
      "external_system": "pdmac-sql",
      "external_key": "docs-tree-set",
      "name": "Laundry",
      "fields": {
        "Species": 7272,
        "Finish": 7340,
        "Sheen": 7347,
        "Interior": 7344,
        "Upper Door Style": 24753,
        "Lower Door Style": 24783,
        "Drawer": 7382,
        "Slide": 7392,
        "Front": 7350,
        "Hinge": 7387
      },
      "children": [
        {
          "type_key": "cab_line",
          "external_system": "pdmac-sql",
          "external_key": "docs-tree-line-1",
          "name": "Sink base",
          "properties": {
            "qty": 1,
            "area": "Laundry",
            "unit_cents": 7001,
            "ext_cents": 7001
          },
          "fields": {
            "product": 9009
          },
          "children": [
            {
              "type_key": "cab_modification",
              "external_system": "pdmac-sql",
              "external_key": "docs-tree-mod-1",
              "name": "Finished end"
            }
          ]
        },
        {
          "type_key": "cab_line",
          "external_system": "pdmac-sql",
          "external_key": "docs-tree-line-2",
          "name": "Base, two drawer",
          "properties": {
            "qty": 1,
            "area": "Laundry",
            "unit_cents": 14003,
            "ext_cents": 14003
          }
        }
      ]
    }
  ]
}

```

```

        "fields": {
            "product": 9010
        }
    },
    {
        "type_key": "cab_install",
        "external_system": "pdmac-sql",
        "external_key": "docs-tree-install",
        "name": "Installation",
        "properties": {
            "qty": 1,
            "override_list_cents": 50000,
            "unit_cents": 50000,
            "ext_cents": 50000
        }
    }
]
}

```

One request, one 201. The answer mirrors the body — every resource's id, whether it was created, and its children in the order you sent them:

From `foc-tree-create.json` — status 201:

```

{
  "resource_id": 24925,
  "created": true,
  "children": [
    {
      "resource_id": 24926,
      "created": true,
      "children": [
        {
          "resource_id": 24927,
          "created": true,
          "children": [
            {
              "resource_id": 24928,
              "created": true,
              "children": []
            }
          ]
        },
        {
          "resource_id": 24929,
          "created": true,
          "children": []
        },
        {
          "resource_id": 24930,
          "created": true,
          "children": []
        },
        {
          "resource_id": 24931,
          "created": true,
          "children": []
        },
        {
          "resource_id": 24932,
          "created": true,
          "children": []
        }
      ]
    },
    {
      "resource_id": 24933,
      "created": true,
      "children": []
    }
  ]
}

```

It is all or nothing. If any part is refused — here line three's product field points at a design choice, not a product — nothing in the request is created, and the answer is that part's own error with `item` saying which part:

From `foc-tree-refused-item.json` — status 422:

```
{
  "error": "accepted_type",
  "message": "This port does not accept resources of type Species.",
  "portId": 7928,
  "endpoint": "source",
  "offendingResourceId": 7272,
  "offendingTypeId": 1404,
  "item": {
    "path": "children[0].children[2]",
    "external_key": "docs-tree-bad-line-3"
  }
}
```

Sending the same body again changes nothing and answers 200 with `created: false` throughout (`foc-tree-replay.json`), so a request that timed out can simply be resent. From here the bid goes on as in §7: freeze, Submit, and so on — those are still one request per move. A field changes as in §5.

5. Change or remove a field; the one-set limit

A field holds one choice. To change it, send `replaceInteractionId` with the connection it replaces — here line two moves to a different product in one call:

From `interaction-replace.json` — status 201:

```
{
  "interactionId": 2739,
  "sourceResourceId": 24914,
  "targetResourceId": 9009,
  "interactionTypeId": 278,
  "contextId": null
}
```

To remove a choice, delete the connection: `DELETE /v1/resource-interactions/<interactionId>` answers 204 with no body (`interaction-delete.json`). Here that removes line one's own Hinge, and the line inherits the set's Hinge again.

A line sits in at most one design set. Putting it in a second is refused, and nothing changes:

From `members-422-containment-cap.json` — status 422:

```
{
  "error": "containment_cap",
  "message": "This item can be inside at most 1 Design Set.",
  "childId": 24913,
  "parentTypeId": 1171,
  "maxParents": 1
}
```

6. Installation and modification lines

An installation line sits on the bid itself; a modification sits under the line it modifies. Both are `find-or-create` plus `members` , like a line:

From `line-install-foc.json` — status 201:

```
{
  "resource_id": 24919,
  "created": true
}
```

A modification sub-line carries its own name and price. It has no field pointing at the price-book modification it came from, and it cannot take that modification's identifier (`sku`): the price book's identifiers are unique, so the attempt is `409 unique_key_taken` (`unique-key-taken-409.json`). Keep the link in your own system.

7. Freeze bottom-up

GET `/v1/state-transitions?resourceId=$SET&fromStatusId=<its statusId>` lists the moves; each has a `transitionId`. The checklist for one move:

```
curl -s -H "Authorization: Bearer $TOKEN" "$BASE/v1/resources/<setId>/requirements?transitionId=
<freezeTransitionId>"
```

From `requirements-read.json` — status 200:

```

{
  "requirements": [
    {
      "requirementId": 43,
      "name": "R-SET-FREEZE – all lines frozen",
      "blocking": "hard",
      "satisfied": false,
      "missing": [
        "All lines frozen"
      ],
      "leaves": [
        {
          "leaf": "members",
          "label": "All lines frozen",
          "caption": "0/5 members at required status",
          "satisfied": false,
          "attestable": false,
          "actionLabel": null,
          "actor": null,
          "actorRoles": [],
          "payloadFields": [],
          "blockedByActor": false,
          "satisfiedEver": false,
          "mayEmitUnvocab": false,
          "memberTotal": 5,
          "memberSatisfied": 0
        }
      ]
    },
    {
      "requirementId": 44,
      "name": "Contents compatible",
      "blocking": "hard",
      "satisfied": true,
      "missing": [],
      "leaves": [
        {
          "leaf": "subtree_compat",
          "label": "No design set settings conflicts",
          "caption": "everything inside fits",
          "satisfied": true,
          "attestable": false,
          "actionLabel": null,
          "actor": null,
          "actorRoles": [],
          "payloadFields": [],
          "blockedByActor": false,
          "satisfiedEver": false,
          "mayEmitUnvocab": false,
          "violations": [],
          "violationCount": 0,
          "subtreeCompatVacuous": false
        }
      ]
    }
  ]
}

```

A set freezes only when every line in it is frozen; asked too early, the answer is 422 with this same checklist (`patch-status-422-requirements.json`). So freeze each line first — the modification under it, then the line:

```

curl -s -X PATCH -H "Authorization: Bearer $TOKEN" -H 'Content-Type: application/json' \
  -d '{"toStatusId":<frozenStatusId>}' "$BASE/v1/resources/<lineId>/status"

```

From `patch-status-200.json` — status 200:

```
{
  "resourceId": 24913,
  "fromStatusId": 1318,
  "statusId": 1319
}
```

Then the installation line and the set. Freezing records a snapshot of the line (`resource.captured` , see [events.md](#); `events-resource-captured.json`). A frozen line is read-only, and there is no way back to Draft: asking is 409 (`patch-status-409-invalid-transition.json`).

8. Submit

The bid's own moves from Draft:

```
curl -s -H "Authorization: Bearer $TOKEN" "$BASE/v1/state-transitions?resourceId=
<bidId>&fromStatusId=<draftStatusId>"
```

From `bid-state-transitions.json` — status 200:

```
{
  "transitions": [
    {
      "transitionId": 818,
      "fromStatusId": 280,
      "toStatusId": 1314,
      "actionLabel": "Submit",
      "autoAdvance": false,
      "toStatusName": "Sent",
      "toStatusPreventsUsage": false,
      "toStatusIsAllocating": false,
      "toStatusSubjectReadOnly": true
    },
    {
      "transitionId": 1271,
      "fromStatusId": 280,
      "toStatusId": 282,
      "actionLabel": "Accept internally",
      "autoAdvance": false,
      "toStatusName": "Accepted",
      "toStatusPreventsUsage": false,
      "toStatusIsAllocating": false,
      "toStatusSubjectReadOnly": true
    }
  ]
}
```

Record the totals your application computed — the `bid.sent` attestation, shown in the [Quickstart](#). It is recorded once: a second is 409 (`attestation-409-not-repeatable.json`). Then check Submit's checklist:

```
curl -s -H "Authorization: Bearer $TOKEN" "$BASE/v1/resources/<bidId>/requirements?transitionId=
<submitTransitionId>"
```

From `bid-requirements-submit.json` — status 200:

```

{
  "requirements": [
    {
      "requirementId": 45,
      "name": "R-BID-SEND-M – all design sets & install lines frozen",
      "blocking": "hard",
      "satisfied": true,
      "missing": [],
      "leaves": [
        {
          "leaf": "members",
          "label": "All design sets frozen",
          "caption": "1/1 members at required status",
          "satisfied": true,
          "attestable": false,
          "actionLabel": null,
          "actor": null,
          "actorRoles": [],
          "payloadFields": [],
          "blockedByActor": false,
          "satisfiedEver": false,
          "mayEmitUnvocab": false,
          "memberTotal": 1,
          "memberSatisfied": 1
        },
        {
          "leaf": "members",
          "label": "All install lines frozen",
          "caption": "1/1 members at required status",
          "satisfied": true,
          "attestable": false,
          "actionLabel": null,
          "actor": null,
          "actorRoles": [],
          "payloadFields": [],
          "blockedByActor": false,
          "satisfiedEver": false,
          "mayEmitUnvocab": false,
          "memberTotal": 1,
          "memberSatisfied": 1
        }
      ]
    },
    {
      "requirementId": 47,
      "name": "R-BID-SEND-LEVEL – price level set",
      "blocking": "hard",
      "satisfied": true,
      "missing": [],
      "leaves": [
        {
          "leaf": "edge",
          "label": "Priced at connection to a Price Level",
          "caption": "present",
          "satisfied": true,
          "attestable": false,
          "actionLabel": null,
          "actor": null,
          "actorRoles": [],
          "payloadFields": [],
          "blockedByActor": false,
          "satisfiedEver": false,
          "mayEmitUnvocab": false
        }
      ]
    }
  ],
}

```

```

{
  "requirementId": 46,
  "name": "R-BID-SEND-E – bid totals recorded",
  "blocking": "hard",
  "satisfied": true,
  "missing": [],
  "leaves": [
    {
      "leaf": "evidence",
      "label": "Bid totals recorded (bid.sent)",
      "caption": "recorded",
      "satisfied": true,
      "attestable": true,
      "actionLabel": "Send bid",
      "eventType": "bid.sent",
      "payloadKey": "total_cents",
      "payloadPredicate": {
        "exists": true
      },
      "actor": null,
      "actorRoles": [],
      "payloadFields": [
        {
          "key": "subtotal_cents",
          "kind": "number",
          "label": "subtotal_cents",
          "required": false
        },
        {
          "key": "discount_cents",
          "kind": "number",
          "label": "discount_cents",
          "required": false
        },
        {
          "key": "delivery_cents",
          "kind": "number",
          "label": "delivery_cents",
          "required": false
        },
        {
          "key": "tax_cents",
          "kind": "number",
          "label": "tax_cents",
          "required": false
        },
        {
          "key": "install_cents",
          "kind": "number",
          "label": "install_cents",
          "required": false
        },
        {
          "key": "total_cents",
          "kind": "number",
          "label": "total_cents",
          "required": false
        },
        {
          "key": "tax_rate_bp",
          "kind": "number",
          "label": "tax_rate_bp",
          "required": false
        }
      ]
    },
    {
      "key": "total_cents",
      "kind": "number",
      "label": "total_cents",
      "required": false
    },
    {
      "key": "tax_rate_bp",
      "kind": "number",
      "label": "tax_rate_bp",
      "required": false
    }
  ],
  "repeatable": false,

```

```

        "blockedByActor": false,
        "satisfiedEver": false,
        "mayEmitUnvocab": false
      }
    ]
  },
  {
    "requirementId": 48,
    "name": "R-BID-SEND-TERMS – payment terms set",
    "blocking": "hard",
    "satisfied": true,
    "missing": [],
    "leaves": [
      {
        "leaf": "edge",
        "label": "Pays on connection to a Payment Terms",
        "caption": "present",
        "satisfied": true,
        "attestable": false,
        "actionLabel": null,
        "actor": null,
        "actorRoles": [],
        "payloadFields": [],
        "blockedByActor": false,
        "satisfiedEver": false,
        "mayEmitUnvocab": false
      }
    ]
  }
]
}

```

Submit checks that the bid holds at least one design set, that every design set and install line is frozen, that the totals are recorded, and that the bid's level and terms are filled. Then move it:

```

curl -s -X PATCH -H "Authorization: Bearer $TOKEN" -H 'Content-Type: application/json' \
  -d '{"toStatusId":<sentStatusId>}' "$BASE/v1/resources/<bidId>/status"

```

From `bid-status-submit-sent.json` — status 200:

```

{
  "resourceId": 24911,
  "fromStatusId": 280,
  "statusId": 1314
}

```

A sent bid is read-only:

From `status-read-only-409.json` — status 409:

```

{
  "error": "status_read_only",
  "message": "Harbor Homes – kitchen is Sent, a read-only status – its properties cannot be changed.",
  "resourceId": 24911,
  "statusId": 1314
}

```

GET `/v1/resources/<bidId>/advancement` answers the bid, its moves from Sent and each move's checklist in one call — see [endpoints.md](#).

9. Accept

The customer's acceptance is the `bid.accepted` attestation:

```
curl -s -X POST -H "Authorization: Bearer $TOKEN" -H 'Content-Type: application/json' \
  -d '{"eventType":"bid.accepted","payload":{}}' "$BASE/v1/resources/<bidId>/attestations"
```

From `bid-attest-accepted.json` — status 201:

```
{
  "eventUlid": "01M5FWMHYZ4Y68TMB3N8AJKJBQ",
  "eventType": "bid.accepted",
  "resourceId": 24911,
  "actorResourceId": 24907
}
```

The bid then moves to Accepted **on its own**, a moment later. Its history shows who did what — the move is ReSync's, the acceptance is your key's:

From `bid-advance-accepted.json` — status 200:

```

{
  "items": [
    {
      "event_ulid": "01M5FWMJ1MDV7XZC4K10MH8AHM",
      "event_type": "status.changed",
      "occurred_at": "2026-10-21T18:47:34.709Z",
      "actor": {
        "resource_id": 5,
        "name": "Popular Design Mill & Cabinet, Inc."
      },
      "display": {
        "from_status_id": 1314,
        "to_status_id": 282,
        "from_status_name": "Sent",
        "to_status_name": "Accepted",
        "text": "Sent → Accepted"
      },
      "producer_id": "advance.worker"
    },
    {
      "event_ulid": "01M5FWMHYZ4Y68TMB3N8AJKJBQ",
      "event_type": "bid.accepted",
      "occurred_at": "2026-10-21T18:47:34.623Z",
      "actor": {
        "resource_id": 24907,
        "name": "bid-app-docs"
      },
      "display": {
        "attested_action_label": "Accept bid"
      },
      "producer_id": "bid-app-docs"
    },
    {
      "event_ulid": "01M5FWMHNVHG3M0ZEG8ZNM1HF5",
      "event_type": "status.changed",
      "occurred_at": "2026-10-21T18:47:34.331Z",
      "actor": {
        "resource_id": 24907,
        "name": "bid-app-docs"
      },
      "display": {
        "from_status_id": 280,
        "to_status_id": 1314,
        "from_status_name": "Draft",
        "to_status_name": "Sent",
        "text": "Draft → Sent"
      },
      "producer_id": "bid-app-docs"
    }
  ],
  "nextCursor": "01M5FWMHNVHG3M0ZEG8ZNM1HF5",
  "afterCursor": "c1MTMwMTMxOS4yMzc2NA"
}

```

10. Convert

Convert's checklist asks for one thing — the bid's customer:

From `bid-requirements-convert.json` — status 200:

```
{
  "requirements": [
    {
      "requirementId": 1531,
      "name": "R-CONVERT-EDGE – customer edge present",
      "blocking": "hard",
      "satisfied": true,
      "missing": [],
      "leaves": [
        {
          "leaf": "edge",
          "label": "Bid for connection to a Customer",
          "caption": "present",
          "satisfied": true,
          "attestable": false,
          "actionLabel": null,
          "actor": null,
          "actorRoles": [],
          "payloadFields": [],
          "blockedByActor": false,
          "satisfiedEver": false,
          "mayEmitUnvocab": false
        }
      ]
    }
  ]
}
```

Converting is **one call**. The move to Converted records the conversion itself — the advancement read names it under that move's records, { "eventType": "bid.converted", "label": "Convert to order" } (bid-advancement-accepted.json):

```
curl -s -X PATCH -H "Authorization: Bearer $TOKEN" -H 'Content-Type: application/json' \
  -d '{"toStatusId":<convertedStatusId>}' "$BASE/v1/resources/<bidId>/status"
```

From `bid-status-convert.json` — status 200:

```
{
  "resourceId": 24911,
  "fromStatusId": 282,
  "statusId": 1315
}
```

Do **not** post `bid.converted` afterwards. It is already on the bid, so a second one is refused and nothing changes:

From `bid-attest-converted-409.json` — status 409:

```
{
  "error": "attestation_not_repeatable",
  "message": "Convert to order is already recorded for Harbor Homes – kitchen in its current status.",
  "resourceId": 24911,
  "eventType": "bid.converted"
}
```

11. The order

ReSync creates the order a moment after the move to Converted. Read the bid until its `derivatives` lists it (`get-derivatives-after-convert.json`), or find it by name:

```
curl -s -H "Authorization: Bearer $TOKEN" "$BASE/v1/resources?
typeKey=cab_order&q=Harbor%20Homes&limit=5"
```

From `list-orders-q.json` — status 200:

```
{
  "resources": [
    {
      "resourceId": "4714",
      "name": "Order — Harbor Homes — 2026-10-26",
      "identifier": null,
      "ownerResourceId": 4,
      "resourceTypeId": "28",
      "resourceTypeName": "Order",
      "isSystem": false,
      "createdAt": "2026-10-27T04:13:31.366Z",
      "statusId": 30,
      "statusName": "Received",
      "statusColor": "#64748b",
      "archivedAt": null,
      "icon": "box"
    }
  ],
  "nextCursor": null
}
```

The order is named once, when it is created — "Order — " plus the bid's customer and the date — and keeps that name. Read it:

From `order-get.json` — status 200:

```

{
  "resourceId": "4714",
  "resourceTypeId": "28",
  "ownerResourceId": "4",
  "name": "Order – Harbor Homes – 2026-10-26",
  "identifier": null,
  "properties": {},
  "statusId": "30",
  "isVerified": false,
  "createdAt": "2026-10-27T04:13:31.366Z",
  "archivedAt": null,
  "derivedFromTypeId": null,
  "resourceTypeName": "Order",
  "propertySchema": {},
  "declaredPropertySchema": {},
  "orgName": "Popular Design Mill & Cabinet, Inc.",
  "ownerOrgId": "3",
  "canvasMode": null,
  "statusName": "Received",
  "statusColorCode": "#64748b",
  "icon": "box",
  "lifecycleId": 7,
  "canvasId": null,
  "derivedFrom": [
    {
      "resourceId": 4704,
      "name": "Harbor Homes – kitchen",
      "typeName": "Bid"
    }
  ],
  "derivatives": [],
  "derivedKeys": [],
  "edges": [],
  "otherEdges": [],
  "confirmedKeys": [],
  "readOnly": null
}

```

What to notice. `derivedFrom` points back at the bid. The order is your organization's, and your key may read it but not change it. Its history shows ReSync created it:

From `order-events.json` — status 200:

```

{
  "items": [
    {
      "event_ulid": "01M5XS0DVMVEXBT6DYWBKX33BA",
      "event_type": "interaction.created",
      "occurred_at": "2026-10-27T04:13:31.380Z",
      "actor": {
        "resource_id": 4,
        "name": "Popular Design Mill & Cabinet, Inc."
      },
      "display": {
        "sourcePortId": null,
        "targetPortId": null,
        "interactionId": 860,
        "sourceResourceId": 4714,
        "targetResourceId": 4704,
        "interactionTypeId": 11,
        "text": "connected Order – Harbor Homes – 2026-10-26 – derives from → Harbor Homes –
kitchen",
        "interactionTypeName": "derives_from",
        "forwardLabel": "derives from",
        "reverseLabel": "source of",
        "sourceName": "Order – Harbor Homes – 2026-10-26",
        "targetName": "Harbor Homes – kitchen"
      },
      "producer_id": "consequence.worker"
    },
    {
      "event_ulid": "01M5XS0DVBRVC8RQXC1D839G7H",
      "event_type": "status.changed",
      "occurred_at": "2026-10-27T04:13:31.372Z",
      "actor": {
        "resource_id": 4,
        "name": "Popular Design Mill & Cabinet, Inc."
      },
      "display": {
        "from_status_id": null,
        "to_status_id": 30,
        "from_status_name": null,
        "to_status_name": "Received",
        "text": "set to Received"
      },
      "producer_id": "consequence.worker"
    },
    {
      "event_ulid": "01M5XS0DVA3MBY3T6JXJ1SB0WT",
      "event_type": "resource.created",
      "occurred_at": "2026-10-27T04:13:31.370Z",
      "actor": {
        "resource_id": 4,
        "name": "Popular Design Mill & Cabinet, Inc."
      },
      "display": {
        "name": "Order – Harbor Homes – 2026-10-26",
        "resource_type_id": 28,
        "derived_from_resource_id": 4704,
        "consequence_rule_id": 1,
        "triggered_by_event_ulid": "01M5XS0DSKHFD0S5ZVYBXZE4RP",
        "text": "* created – Order"
      },
      "producer_id": "consequence.worker"
    }
  ],
  "nextCursor": null,

```

```
"afterCursor": "c1MTg2NTMxNS40Njky"
}
```

The order starts **empty** — the bid's lines are not copied into it:

From `order-members-empty.json` — status 200:

```
[]
```

12. Accepted internally, by a named person

Some bids are accepted inside your organization rather than by the customer: `bid.accepted_internally`, naming the person who approved it in `actorResourceId`. That person must belong to your organization; anyone else is `403 actor_not_readable` (`attestation-403-actor-not-readable.json`), and nothing is recorded.

Like Convert, accepting internally needs the bid's **customer** field (\$2) — for a bid sold inside your organization, the internal customer. Without it the move to Accepted is `422`, naming "R-ACCEPT-INTERNAL-EDGE — customer edge present" among its `requirements`, and the bid stays Draft.

From `bid-attest-accepted-internally.json` — status 201:

```
{
  "eventUlid": "01M5FWMM2YF3QKXSW8RHBE4XJS",
  "eventType": "bid.accepted_internally",
  "resourceId": 24922,
  "actorResourceId": 24905
}
```

Things to know

1. **Every resource you create through `find-or-create` binds one identifier in your external system** — whether it came alone or inside a bigger body. A bid with one design set and five lines is seven identifiers; the customer is one more.
2. **A second `find-or-create` with the same identifier changes nothing.** It answers `200` with the existing resource and ignores the `name` and `properties` you sent. To change a resource, `PATCH` it. A body with `children` replays the same way, resource by resource; a field that already points somewhere else is refused, not changed.
3. **A `PATCH` merges every key you send, declared or not.** A property the type does not declare is stored as sent, without a check. Send only the keys the type declares (its `propertySchema`, [Quickstart](#)).
4. **Saving a new bid is one request; the rest of its life is step by step.** `find-or-create` with `fields` and `children` writes the bid, its design set, every line and their fields at once ([above](#); up to 250 resources per request). Saved step by step instead, each line is about 8 requests — create, put it in its set, set its prices, and its product field. Either way the bid's later moves (freezes, Submit, accept, convert) are one request each, and each line's freeze is one request per line.
5. **The first sync is about 66 requests.** Reading the price book, the price levels, the payment terms, the design choices, the customers and the reference lists at `limit=100` takes about 66 requests — about 5% of a 1,200-per-minute window, or a fifth of the default 300.

6. **The rate limit is 300 requests per minute per key by default**, counted in a fixed one-minute window that starts with your first request. ReSync can set a different limit for your key — ask your administrator; your key's own limit is in every response's `x-ratelimit-limit`. On `429`, wait `retry-after` seconds and continue.
7. **Member rows carry no status**. Reading a tree's statuses costs one `GET` per child.
8. **A freeze or a Submit records a snapshot** (`resource.captured`). Your key sees the snapshot's own property values; the settings and the product it photographed are kept on the record but are not passed to a key.
9. **Who approves**. An attestation or status move is attributed to your key's own service identity unless you name a person: `actorResourceId` may name any person of your organization; someone outside it is `403 actor_not_readable`.
10. **Identifiers are unique per external system across every organization**, not only yours. Prefix them with something that is yours.
11. **Freezing is one-way, and a sent bid is final**. Nothing leaves Frozen (`409 Invalid status transition`). A revision is a new bid; linking it to the one it replaces is not yet available to keys (`interaction-400-ports-required.json`).
12. **The order is created empty**. Its lines are not copied from the bid; read them from the bid.
13. **A modification sub-line does not point at its price-book modification**. It carries its own name and price; keep the link in your own system.
14. **Compatibility is answered per field**. Legal-targets gives each candidate's verdict and reasons; a key has no bulk read of the compatibility rules.

Where to go next

- [endpoints.md](#) — every door, with parameters and errors.
- [errors.md](#) — every error code and what to do about it.
- [events.md](#) — what each event in a resource's history means.
- [conventions.md](#) — the rules that apply everywhere, and what you cannot do yet.

4 Authentication and scopes

Sending the key

Every request carries the key as a bearer token:

```
Authorization: Bearer <your key>
```

There is no other accepted form — no query parameter, no custom header, no cookie. The key is shown once, when it is created. It is not recoverable afterwards; if it is lost, a new one is issued.

People sign in; apps hold credentials

A **key** (a credential) is your app's own identity, for work no person is doing. When a **person** uses your app, they can sign in with their ReSync account instead, and your app then acts as them, with their access and never more than the app is registered for — see [sign-in.md](#). Both are sent the same way, as the bearer token.

What a key is bound to

A key belongs to exactly one organization, called its **owner**. That binding is fixed for the life of the key: it cannot be moved to another organization, and a key cannot be made to act for two.

Everything the key can reach is owned by that organization. There is no shared or public pool of resources, so a resource belonging to another organization is refused whether you reach for it by id, by list, or through a related record.

The scope sets

A key's scopes are lists. Three type sets — `read_types`, `operate_types` and `attest_types` — govern most doors (the table below). `find_or_create_types` is a fourth type set that governs one door, and `bind_systems` lists external systems, not types.

Set	Grants	Applies to
<code>read_types</code>	list and fetch	<code>GET</code> doors
<code>operate_types</code>	change	<code>PATCH</code> , <code>enrol</code> , <code>exit</code> , <code>attest</code> , <code>status</code>
<code>attest_types</code>	record and advance, never change	<code>attest</code> , <code>status</code> — and the <code>GET</code> doors

Operating implies reading, and so does attesting. A type in `operate_types` or `attest_types` does not need repeating in `read_types`; the readable set is the union of the three. `attest_types` lets a key record attestations and move status on a type without being able to edit, enrol or exit anything of that type (see [Attesting without changing](#)).

Which endpoints need which:

- **Read scope** — GET `/v1/resources` (list), GET `/v1/resources/:id`, GET `/v1/resources/search`, and the per-resource reads: `/events`, `/advancement`, `/members`, `/containers`, `/capabilities`, `/requirements`, `/ports`, `/ports/:portId/legal-targets`.
- **Attest scope** (`attest_types` or `operate_types`) — POST `/v1/resources/:id/attestations` and PATCH `/v1/resources/:id/status`.
- **Operate scope** — PATCH `/v1/resources/:id`, POST `/v1/resources/:id/members`, POST `/v1/resources/:id/exit`, POST `/v1/resources/:id/move`, POST `/v1/resources/:id/attestations`, PATCH `/v1/resources/:id/status`, the `enrollInto` / `moveOf` / `moveFrom` filters of search, and POST / DELETE `/v1/resource-interactions` — operate on the side whose port points **out** (the side making the choice), read on the side it points to.
- **bind_systems** — the external systems your key may bind identifiers under: POST `/v1/aliases/bind` and POST `/v1/identity-claims/redeem`.

`find_or_create_types` governs POST `/v1/resources/find-or-create` only. It is separate on purpose: being able to change a bid is not the same as being able to bring one into existence.

The general-reference lists — types, statuses, lifecycles, units and so on — are not type-scoped. Any valid key may read them. A by-id read of one of them (a type, a lifecycle, its graph, its transitions) answers 404 for anything the key's own list would not show.

POST `/v1/ingest` also authenticates an API key. Its contract is not yet part of this pack.

Attesting without changing

`attest_types` is a list of types on which the key may record attestations and move status, and nothing more. It reads them too. It does not let the key edit, enrol or exit: those still need `operate_types`, and a key without it is refused 403 `operate_types_required`. Use it for a client whose job is to report what happened — a scanner, a checklist app — rather than to change records.

Device credentials

A **device credential** is a key that acts at one resource — a workstation, a counter, a door. An administrator at the owning organization issues it like any key, with up to three more scope entries:

Scope	Meaning
<code>acts_at_resource_id</code>	The resource the device acts at. The key may read it and everything inside it, and record attestations and move status there. It never edits, enrolls or exits.
<code>actor_source</code>	<code>presence_session</code> — the device acts for whoever has checked in at it, never for itself.
<code>presence_session_ttl_s</code>	Seconds of inactivity before a check-in ends: 30–3600, 300 when absent.

A device credential may also hold the other scope sets; `attest_types` is the usual one, for types it records on outside its own resource.

Checking people in. A device credential with `actor_source` opens a **presence session** when a person presents an identifier to the device ([endpoints](#)): `POST /v1/presence-sessions` with the identifier's system and code. One person is checked in at a time; a different person's check-in ends the previous one. The session ends on its own after `presence_session_ttl_s` seconds without activity; every write made under it, and `POST /v1/presence-sessions/current/touch`, counts as activity.

Acting for the person. Every write by a device credential with `actor_source` must carry the `presenceSessionId` it received at check-in. The write is recorded as that person's act. It cannot name an actor any other way: `actorResourceId` and `actorAlias` are refused (`403 actor_source_presence_session_only`), a missing id is `400 presence_session_required`, and an ended or expired session is `401 presence_session_expired` — check the person in again. The id is returned as a string; a string or a number is accepted back.

Linking an identifier. An administrator issues a one-time enrolment code for a person in ReSync; the person presents the code and the identifier to the device, which calls `POST /v1/alias-enrollments/redeem`. The enrolment must belong to the key's own organization.

Live updates. A device credential may hold the live-update stream, `GET /v1/live` — for its own resource and what it can read. Any other key is refused (below).

What it checks every time. On every request a device credential's resource must still exist (`403 acts_at_resource_missing`) and still belong to the key's organization (`403 acts_at_outside_owner_grant`).

`GET /v1/api-keys/self` returns these entries inside `scopes`, and `actsAt` names the resource:

From `device-self.json` — status 200:

```
{
  "apiKeyId": 785,
  "name": "device-capture",
  "scopes": {
    "actor_source": "presence_session",
    "attest_types": [
      "person"
    ],
    "acts_at_resource_id": 24914,
    "presence_session_ttl_s": 300,
    "created_owner_resource_id": 5
  },
  "actsAt": {
    "resourceId": 24914,
    "name": "Capture Station"
  },
  "owner": {
    "resourceId": 5,
    "name": "Popular Design Mill & Cabinet, Inc."
  }
}
```

Telling ReSync which device. A device credential's app may send `x-device-id` on its requests — an id the app makes once per install (letters, digits, `.`, `,`, `_`, `:` and `-`; up to 128 characters). The first request from a new device is recorded on the credential's history with the device's `user-agent`, so an administrator can see where a credential is in use. Nothing else changes; a request without the header works the same.

Changing scopes

You cannot change your own scopes. An administrator at the owning organization does it, through a surface your key cannot reach. Ask them; do not build a retry around a `403`.

Every scope change is recorded on the ledger, with the complete before and after. So is the creation of the key and its revocation. If you need to know why a call that worked last week fails today, that record is the answer, and your administrator can read it.

Revocation

A key is revoked, not deleted. Once revoked it stops authenticating immediately and permanently; there is no un-revoke. Everything it wrote stays on the ledger and stays attributed to it.

Keys do not expire on their own. A key is valid until somebody revokes it — so treat it like a password that never rotates itself, and ask for a new one if you suspect it has leaked.

A machine key that belongs to a connected app ([sign-in.md](#)) is rotated by the organization's administrator from the app's page: a new key under a new name is minted and shown once, then the old key is revoked. Revoking the app revokes every key that belongs to it.

Rate limit

300 requests per minute per key by default. The budget is counted per credential, so another integration cannot spend yours and you cannot spend theirs. ReSync may set a different limit for your key — for a large integration, ask your administrator; it is not something a key can change for itself. Whatever your key's limit is, every response says so:

```
x-ratelimit-limit: 300
x-ratelimit-remaining: 297
x-ratelimit-reset: 59
```

The window is **fixed**, not rolling: it starts with your first request and lasts one minute; when it ends, the whole budget is back. `x-ratelimit-reset` is the seconds left in the current window.

When you exhaust the budget the answer is `429`, with `retry-after` (seconds left in the window) beside the same headers. Captured on a key whose own limit is 3, at its fourth request:

```
x-ratelimit-limit: 3
x-ratelimit-remaining: 0
x-ratelimit-reset: 60
retry-after: 60
```

From `rate-limited-429.json` — status 429:

```
{
  "error": "rate_limited",
  "message": "rate limit exceeded – 3 requests per 1 minute",
  "retryAfterSeconds": 60
}
```

`message` reads "... requests per "; `retryAfterSeconds` is the same wait as `retry-after`. Honour it: wait, then continue. Do not retry immediately, and do not spread the same work across several keys to get around the limit — ask for a higher one instead.

What a key cannot do yet

Hold a live stream. `GET /v1/live` — the live-update stream ReSync's web app and device credentials use — refuses an integration key (device credentials may hold it). Poll instead ([events.md](#)).

From `live-key-not-supported.json` — status 403:

```
{
  "error": "live_key_not_supported",
  "message": "Live updates are available to sessions and device credentials only; poll the events read with ?after= instead"
}
```

Request ids

Every response carries `x-request-id`.

If you send one, it is echoed back, so you can use your own correlation id end to end:

```
curl -s -H "Authorization: Bearer $TOKEN" -H "x-request-id: order-sync-8842"
"$BASE/v1/resources/<customerId>"
```

Ids are sanitised: letters, digits, `.`, `_`, `:` and `-` survive, everything else is stripped, and the result is truncated to 64 characters. If you send nothing, one is generated for you.

Quote it when you report a problem. It is how a specific request of yours is found in the server logs.

5 Sign in with ReSync

People sign in to your app with their ReSync account. Your app gets a token that **is** the bearer the API accepts: it acts for that person, with that person's access — and never more than your app was registered for.

This is standard **OpenID Connect**: the authorization code flow with **PKCE (S256)**, nothing else. Use a mainstream OIDC client library; you need two values to configure it — the **issuer** and your **client id**.

Issuer	<code>https://auth.limnal.dev</code>
Discovery	<code>https://auth.limnal.dev/.well-known/openid-configuration</code>
Your client id	given to you when your app is registered (see Getting registered)

Your app never asks for or stores a password and has no login form of its own: the person signs in on ReSync's own pages. There is no exchange step and no second call — the access token you receive is the bearer token for the API.

Getting registered

An app is registered by an **administrator of the organization that owns it**: in ReSync they open the organization's **Access** tab, choose **Connected apps** → **Register app**, and from then on manage its redirect URIs, allowed types and machine keys on the app's own page. Send them the **exact https redirect URI** your app will use (scheme, host, port, path — character for character); they give you back the **client id**. There is no self-service registration in this version: an app owned by another organization is registered by that organization's administrator. A registration holds:

Name	what people see on the consent screen
Redirect URIs	where ReSync may send the browser back. Matched exactly — scheme, host, port, path and query. https only.
Allowed types	the types your app may see , attest on, or change — for any person who signs in
Kind	<code>public</code> : your app holds no secret. Confidential clients (a server-side secret at the token endpoint) are not offered yet .

An administrator can **suspend** your app (sign-in and every token stop working, and it can be reactivated) or **revoke** it (permanent). Either takes effect at sign-in at once and for tokens already issued within a minute.

Discovery

Read the endpoints from discovery; do not hard-code them.

From `signin-discovery.json` — status 200:

```
{
  "issuer": "https://auth.limnal.dev",
  "authorization_endpoint": "https://app.limnal.dev/authorize",
  "token_endpoint": "https://auth.limnal.dev/oauth/token",
  "userinfo_endpoint": "https://auth.limnal.dev/oauth/userinfo",
  "jwks_uri": "https://auth.limnal.dev/.well-known/jwks.json",
  "response_types_supported": [
    "code"
  ],
  "grant_types_supported": [
    "authorization_code"
  ],
  "code_challenge_methods_supported": [
    "S256"
  ],
  "token_endpoint_auth_methods_supported": [
    "none"
  ],
  "id_token_signing_alg_values_supported": [
    "ES256"
  ],
  "subject_types_supported": [
    "public"
  ],
  "scopes_supported": [
    "openid",
    "profile"
  ],
  "claims_supported": [
    "sub",
    "iss",
    "aud",
    "exp",
    "iat",
    "name",
    "scope"
  ]
}
```

The signing keys (`jwks_uri`) carry only public members. During a key rotation both keys are listed; pick by `kid` .

From `signin-jwks.json` — status 200:

```
{
  "keys": [
    {
      "kty": "EC",
      "crv": "P-256",
      "x": "vQRDXP46elARbJlG7xsd-09cq5zd3VsRw5vnYx0-5W4",
      "y": "Hydvax4BwK8zwgBZVqFg3vL4euyD0Nt9f0QRKz648tE",
      "kid": "mjowhmaooAQ-51NL8mr_BTpCMUSsYh1kiwF0ZaDugIw",
      "alg": "ES256",
      "use": "sig"
    }
  ]
}
```

The flow

1. **Your app** makes a PKCE pair (a random `code_verifier`, and `code_challenge` = BASE64URL(SHA-256(verifier))), a random `state`, and a `nonce`; it sends the browser to the `authorization_endpoint`:

Parameter	
<code>response_type</code>	<code>code</code>
<code>client_id</code>	your client id
<code>redirect_uri</code>	one of your registered redirect URIs, exactly
<code>scope</code>	<code>openid</code> (required), plus <code>profile</code> for the person's name. Nothing else is supported.
<code>state</code>	1–512 characters; returned to you unchanged
<code>code_challenge</code>	43–128 base64url characters
<code>code_challenge_method</code>	S256
<code>nonce</code>	optional, up to 255 characters; returned in the ID token

2. **The person** signs in to ReSync if they are not already, and sees a consent screen: your app's name, its organization, and what it can see, attest and change. A request that is malformed, for an unknown or inactive app, or with a redirect URI your app did not register is refused **on that screen** — ReSync never sends the browser to a redirect URI it has not validated.
3. **Allow** → the browser returns to `redirect_uri?code=...&state=...`. **Deny** → `redirect_uri?error=access_denied&state=...`. Check `state` before you use the code.
4. **Your app** exchanges the code (once, within **60 seconds**) at the `token_endpoint`, with the verifier:

```
POST https://auth.limnal.dev/oauth/token
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&code=...&redirect_uri=...&client_id=...&code_verifier=...
```

A JSON body with the same fields is accepted too.

From `signin-token.json` — status 200:

```
{
  "access_token":
"eyJhbGciOiJIUzI1NiIsImtpZCI6Im1qb3dobWFvbn0FRlTUxTkW4bXJfQlRwQ001VU3NZaDFraXdGT1phRHVnSXciLCJ0eXAiOiJhdCtqd3QifQ.eyJjbGllbnRfaWQiOiIwMU01Vf03RVBSUjltWTQ3NzNNV0Y0RUZwUSIsInNjb3BlIjoib3BlbmIiIHByb2ZpbGUiLCJpc3MiOiJodHRwczovL2FldGgubGltbmFsLmRldiIsInN1YiI6IjI0OTA5IiwiaXVkiOiMDfNnVRa0VQUlI5Vk00NzczTVdGNEVGVLiEiLCJpYXQiOiJlE30TI5ODAyNzQsImV4cCI6MTc5Mjk4Mzg3NCwianRpIjoim1BTZ2MwamQ2UGVVM0dWUTBrcWVZyJ9.xZduW45HAD_YDgkfc3NbUaQF2Sn0m8xZiYPF1vp2Z9Re6tMHSDR-0kM10re4UACQLRY_yPoBstoi_3emXSIK5w",
  "token_type": "Bearer",
  "expires_in": 3600,
  "id_token":
"eyJhbGciOiJIUzI1NiIsImtpZCI6Im1qb3dobWFvbn0FRlTUxTkW4bXJfQlRwQ001VU3NZaDFraXdGT1phRHVnSXciLCJ0eXAiOiJKV1QiIjE3ODAyNzQsIm5vbmNlIjoib3BlbmFtZSI6ImlpZ24taW4gY2FwdHVsZSBwZXJzb24iLCJpc3MiOiJodHRwczovL2FldGgubGltbmFsLmRldiIsInN1YiI6IjI0OTA5IiwiaXVkiOiMDfNnVRa0VQUlI5Vk00NzczTVdGNEVGVLiEiLCJpYXQiOiJlE30TI5ODAyNzQsImV4cCI6MTc5Mjk4Mzg3NH0.UX2eLSQFoT-4nkHBNtwhgq-0wipnUnrvx1KINTH-_arWy2Sd-dFFomk1jV_G5Pmh8rdSu6G0K5DCky5g4Ss5MA",
  "scope": "openid profile"
}
```

The answer is never cached (`Cache-Control: no-store`). A code works **once**: a second exchange, a wrong verifier, a different `client_id` or `redirect_uri` all answer `invalid_grant` — and a failed exchange still uses the code up.

From `signin-token-invalid-grant.json` — status 400:

```
{
  "error": "invalid_grant",
  "error_description": "The authorization code is invalid, expired or already used."
}
```

The tokens

Both are JWTs signed with **ES256**; verify them against the discovery `jwks_uri`.

Claim	Access token (<code>typ: at+jwt</code>)	ID token (<code>typ: JWT</code>)
<code>iss</code>	<code>https://auth.liminal.dev</code>	the same
<code>sub</code>	the person's ReSync id, as a string	the same
<code>aud</code>	your client id	the same
<code>client_id</code>	your client id	—
<code>scope</code>	the scopes granted	—
<code>jti</code>	a unique id	—
<code>iat</code> , <code>exp</code>	issued at; expires one hour later	the same
<code>auth_time</code>	—	when the person signed in
<code>nonce</code>	—	the <code>nonce</code> you sent
<code>name</code>	—	the person's name, with <code>profile</code>

`sub` is stable for life — it does not change when the person's email, name or sign-in method changes. Link your records to `sub`, never to an email; cache only `sub` and a display name. No email address is shared in this version.

There are no refresh tokens. When the access token expires, run the flow again: while the person's ReSync sign-in is still current it completes without asking them anything. Treat "token expired" as one event in your app.

Calling the API

Send the access token exactly as you would a key:

```
Authorization: Bearer <access token>
```

The same endpoints, request shapes and error codes as with a key. The writes your app makes are recorded as made by the person, through your app.

Userinfo. GET `https://auth.limnal.dev/oauth/userinfo` with the access token answers who the person is, from ReSync data only:

From `signin-userinfo.json` — status 200:

```
{
  "sub": "24909",
  "name": "Sign-in capture person"
}
```

What an app token may do

An app signed in for a person may use the **resource** endpoints (reading, editing, members, exits, moves, attestations, status, connections), the **edge** and **canvas** endpoints, **live updates**, the **reference** reads (types, statuses, lifecycles and the rest of the model, read-only), GET `/v1/orgs` (the person's organizations) and GET `/v1/persons/me`. Every other endpoint — messages, keys, access administration, the person's own settings — answers:

From `signin-client-not-permitted.json` — status 403:

```
{
  "error": "client_not_permitted",
  "message": "An app signed in for a person cannot use this door"
}
```

Two permissions, both required. Every call needs the **person's** access (what they can reach and do in ReSync) **and** your **app's** registration (the types it may see, attest or change). Neither widens the other.

Refused because	Code	Remedy
the type is not among the app's types at all	403 type_not_readable	the app's registration
the app may see the type but not attest or change it	403 type_not_operable	the app's registration
the app may change nothing at all	403 operate_types_required	the app's registration
the person's own access does not include the action	403 verb_not_granted	the person's access, in ReSync
the resource is outside everything the person can reach	403 subject_outside_grant	none — that is the boundary
the person's personal space is not open for changes	403 personal_modeling_off	none

A 403 is final for that call: do not retry it, and do not duplicate these rules in your app — ask, and show the answer. The model itself (types, lifecycles) is readable to your app and never changeable through it. Every code is listed in [errors.md](#).

Errors on the two OAuth endpoints

`/oauth/token` and `/oauth/userinfo` answer in the **standard OAuth 2.0 / OpenID Connect shape**, because client libraries expect it — the one place this API does not use its own error envelope:

```
{ "error": "invalid_grant", "error_description": "The authorization code is invalid, expired or already used." }
```

Code	Endpoint	Meaning
<code>invalid_request</code>	token	a parameter is missing or malformed
<code>unsupported_grant_type</code>	token	only <code>authorization_code</code> is supported
<code>invalid_grant</code>	token	the code is unknown, expired, used, or does not match the verifier, client or redirect URI
<code>invalid_client</code>	token	the app is not active
<code>invalid_token</code>	userinfo	the access token is invalid or expired, or its app is not active (401 , with <code>WWW-Authenticate: Bearer error="invalid_token"</code>)

On every `/v1` endpoint, an app token whose app has been suspended or revoked answers 401 `client_not_active`.

Keys and sign-in together

Keep your app's **key** for work no person is doing — imports, syncs, scheduled jobs. Use the **sign-in token** for what a person does in your app, so it is done as them. Build one API client that takes a bearer, and one small sign-in interface — `signIn()`, `signOut()`, `currentUser()` returning `{ id, displayName }` (from `sub` and `name`) — so the rest of your app does not care which it holds.

The two endpoints behind the consent screen

ReSync's own consent screen uses two endpoints with the person's ReSync session — never with an app token or a key (403 `client_not_permitted` / 403 `session_required`). They are listed in [endpoints.md](#) for completeness; your app does not call them.

6 Endpoints

Every door a key can open, and nothing else. Endpoints that exist but need a signed-in person are not listed — they will return `401` or `403` for you no matter what you send. The doors only a device credential can use are under [Device credentials](#).

All paths are relative to `$BASE`. All request and response bodies are JSON.

Pagination

Two endpoints page, and both work the same way:

1. Call with `limit` (1–100).
2. Read `nextCursor` from the response.
3. If it is a string, pass it back as `cursor` for the next page.
4. If it is `null`, you have the last page.

Cursors are **opaque**. Do not parse, compare or construct them; the only valid cursor is one the API just gave you. A full last page returns a non-null cursor, so the final request of a walk returns an empty `items / resources` array — that is normal, not an error.

Discovery

`GET /v1/api-keys/self`

What this key is and what it may touch. Takes no parameters. Every valid key may call it.

Field	Type	Meaning
<code>apiKeyId</code>	number	This key's id.
<code>name</code>	string	This key's name. It is also the producer recorded on everything you write, including the resources you create.
<code>scopes</code>	object	The raw scope sets: <code>read_types</code> , <code>operate_types</code> , <code>attest_types</code> , <code>find_or_create_types</code> , <code>bind_systems</code> — and, on a device credential, <code>acts_at_resource_id</code> , <code>actor_source</code> , <code>presence_session_ttl_s</code> (auth-and-scopes.md). Only the sets the key holds appear.
<code>owner</code>	object null	<code>{ resourceId, name }</code> — the organization that owns the key.
<code>actsAt</code>	object null	<code>{ resourceId, name }</code> — the resource a device credential acts at. Always <code>null</code> for any other key.

Errors: 401 invalid_api_key.

From self.json — status 200:

```

{
  "apiKeyId": 783,
  "name": "bid-app-docs",
  "scopes": {
    "read_types": [
      "cab_pricing_level",
      "cab_payment_terms",
      "cab_order",
      "cab_catalog_product",
      "cab_cat_accessory",
      "cab_cat_base",
      "cab_cat_molding",
      "cab_cat_office",
      "cab_cat_utility",
      "cab_cat_vanity",
      "cab_cat_wall",
      "cab_fam_base_sink",
      "cab_fam_bookcase",
      "cab_fam_crown_molding",
      "cab_fam_filler",
      "cab_fam_lazy_susan",
      "cab_fam_oven_cabinet",
      "cab_fam_panel",
      "cab_fam_post",
      "cab_fam_utility_cabinet",
      "cab_fam_vanity_combo",
      "cab_fam_wall_cabinet",
      "cab_fam_wall_drawer",
      "cab_fam_wall_pantry_pullout_filler",
      "cab_fam_wall_stacked",
      "door_style_lower",
      "door_style_upper",
      "drawer",
      "finish",
      "front",
      "hinge",
      "interior",
      "sheen",
      "slide",
      "species"
    ],
    "operate_types": [
      "cab_customer",
      "cab_bid",
      "cab_design_set",
      "cab_line",
      "cab_install",
      "cab_adjustment",
      "cab_modification"
    ],
    "find_or_create_types": [
      "cab_customer",
      "cab_bid",
      "cab_design_set",
      "cab_line",
      "cab_install",
      "cab_adjustment",
      "cab_modification"
    ],
    "created_owner_resource_id": 5
  },
  "actsAt": null,
  "owner": {
    "resourceId": 5,
    "name": "Popular Design Mill & Cabinet, Inc."
  }
}

```

```
}  
}
```

GET /v1/api-keys/self/view

The resource your key **acts at**, as its **default view** shows it. Takes no parameters — the key names the resource (**actsAt** in GET /v1/api-keys/self); a key that acts at no resource gets **404 no_acting_resource** . Your key must be able to read that resource; if it can, it can read the resource's default view.

The body is the resource's graph — the resource itself, the members and connected resources your key can read, the connections between them, a per-node summary, and the stored view's layout — plus one field. A device credential's resource with no default view, as its key reads it:

From `device-view.json` — status 200:

```

{
  "nodes": [
    {
      "id": "4700",
      "parentId": null,
      "resourceId": "4700",
      "name": "Capture Station",
      "statusName": null,
      "statusColorCode": null,
      "statusAttention": null,
      "statusShape": null,
      "resourceTypeId": 15,
      "derivedFromTypeId": null,
      "resourceTypeName": "Space",
      "isPlaceholder": false,
      "slotId": null,
      "readable": true,
      "memberCount": 0,
      "icon": "box"
    }
  ],
  "context": {
    "resourceId": "4700",
    "name": "Capture Station",
    "statusName": null,
    "statusColorCode": null
  },
  "interactions": [],
  "alerts": [],
  "tray": {
    "noInteractions": 0,
    "hidden": 0
  },
  "obligations": [],
  "summary": {
    "allowedInteractionTypes": [
      {
        "id": 1,
        "name": "allocates"
      },
      {
        "id": 2,
        "name": "consumes"
      },
      {
        "id": 3,
        "name": "reserves"
      },
      {
        "id": 4,
        "name": "produces"
      },
      {
        "id": 5,
        "name": "depends_on"
      },
      {
        "id": 6,
        "name": "assigned_to"
      },
      {
        "id": 7,
        "name": "certified_in"
      },
      {
        "id": 8,

```

```

    "name": "licensed_for"
  },
  {
    "id": 11,
    "name": "derives_from"
  },
  {
    "id": 12,
    "name": "feeds"
  },
  {
    "id": 13,
    "name": "bid_for"
  },
  {
    "id": 14,
    "name": "priced_at"
  },
  {
    "id": 15,
    "name": "pays_on"
  },
  {
    "id": 16,
    "name": "uses"
  },
  {
    "id": 17,
    "name": "sells"
  }
],
"types": {
  "15": {
    "declaredSchema": {}
  }
},
"transitionsByStatus": {},
"slotTypes": {},
"nodes": {
  "4700": {
    "properties": {},
    "derivedKeys": [],
    "transitionKey": null,
    "ports": []
  }
}
},
"view": null
}

```

The fields:

Field	Type	Meaning
nodes	array	Every node drawn: the resource, its readable members, and resources connected to them. A node your key cannot read appears only as <code>{ id, readable: false }</code> . Each readable node carries its status's <code>statusAttention</code> (<code>none</code> · <code>watch</code> · <code>alert</code>) and <code>statusShape</code> (<code>circle</code> · <code>square</code> · <code>triangle</code> · <code>diamond</code> , or <code>null</code> — then draw the shape for its attention: <code>none</code> → <code>circle</code> , <code>watch</code> → <code>triangle</code> , <code>alert</code> → <code>diamond</code>), and <code>icon</code> : its type's icon (see <code>GET /v1/resources</code>). Nodes are in list order : the resource itself first, then nodes whose status attention is <code>alert</code> , then <code>watch</code> , then the rest — within each group, members first (in the order the graph reads them), then outside nodes. List them as given.
interactions	array	The connections between drawn nodes. Field attachments (a value set on a field port) are not drawn as interactions — they appear as port settings on the node summary (<code>summary.nodes[id].ports[].setting</code>), and a value reachable only through one is not a node.
context	object	<code>{ resourceId, name, statusName, statusColorCode }</code> for the resource itself.
alerts	array	The statuses that ask for attention among the resource's readable members — every member, including ones too many to draw: <code>{ statusName, colorCode, attention, shape, count, sampleResourceIds }</code> , <code>watch</code> / <code>alert</code> only, alerts first. Empty when nothing needs attention. Under a type filter, only members the filter keeps count.
summary	object	Per-node detail for the drawn nodes (properties, derived keys, available transitions, ports). A type with a presentation profile carries <code>types[id].profile = { icon, keyProperties, grammar }</code> — <code>grammar</code> is how the canvas lays out a resource of that type, <code>flow</code> or <code>sheet</code> (never null; a type with no <code>profile</code> lays out as <code>flow</code>); a node of that type carries <code>keys</code> (just those properties' values). A node that inherits a field value from an enclosing item carries <code>inherited = [{ kind: { resourceId, typeName }, value, source, locked }]</code> (for kinds it has no field of its own; <code>value</code> / <code>source</code> shaped like a port <code>setting</code> 's). The icon to draw is the node's own <code>icon</code> .
overlay	object	Present when the resource has a default view: its stored positions and its settings. <code>overlay.nodes</code> holds a position row only for a node in <code>nodes</code> — never for a node the view hides or leaves out.
tray	object	<code>{ noInteractions, hidden }</code> : how many of the resource's readable members have no interaction at all (including ones too many to draw; under a type filter, only kept types), and how many nodes the default view hides.
view	object null	<code>{ name, settings }</code> of the default view; <code>null</code> when the resource has none — then the plain graph is returned (not an error). <code>settings = { defaultMode, statusColor, highlightTypeId, filterTypeIds, density, showLogic }</code> .

The view's settings are already applied to the body: a type filter has removed the nodes it leaves out (counted in the resource node's `rollup.filtered`), and the nodes the view hides are left out with their connections (counted in `tray.hidden`; they still count in `alerts`, `memberCount` and the rollup). Each drawn member and outside node carries `connected` (whether it has any interaction). Highlight, status colour, density and `showLogic` (whether the view draws its logic layer) are for your renderer. A name your key cannot read is never sent. Type names on this door follow what the person who provisioned your key can read; a type they cannot read reads "a type".

Errors: `401 invalid_api_key` · `404 no_acting_resource` · the key-arm read refusals of `GET /v1/resources/:id`.

A key that acts at no resource:

From `device-view-no-acting.json` — status 404:

```
{
  "error": "no_acting_resource",
  "message": "This key does not act at a resource."
}
```

Device credentials

The doors only a **device credential** can use — a key that acts at one resource ([auth-and-scopes.md](#)). Ids on these doors are strings. Any other key is refused 403 `presence_scopes_required`; a missing, unknown or revoked key 401 `invalid_api_key`. On every call the credential's resource must still exist and still belong to its organization (403 `acts_at_resource_missing`, 403 `acts_at_outside_owner_grant`).

POST /v1/presence-sessions

Check a person in at the device's resource, by an identifier presented to the device. Needs `actor_source`.

Field	Type	Meaning
<code>system</code>	string	The identifier's system, by name (for example <code>badge</code>).
<code>code</code>	string	The identifier as read. It is normalized by the system's own rule before it is looked up.

Answers 201 with the new check-in, or 200 with `renewed: true` when the same person checks in again (the session is extended; nothing is recorded). A different person's check-in ends the previous one. The answer:

Field	Type	Meaning
<code>presenceSessionId</code>	string	Send it as <code>presenceSessionId</code> on every write made for this person.
<code>person</code>	object	{ <code>resourceId</code> , <code>name</code> } — who checked in.
<code>expiresAt</code>	string	When the session ends without activity (ISO 8601). Every write under it, and <code>touch</code> , extends it.
<code>actsAt</code>	object	{ <code>resourceId</code> , <code>name</code> } — the device's resource.
<code>viaSystem</code>	string	The identifier's system.
<code>renewed</code>	boolean	<code>true</code> when this extended an existing session for the same person.

Records `presence.checked_in` on the device's resource, with the person as the actor ([events.md](#)).

From `presence-create.json` — status 201:

```
{
  "presenceSessionId": "977",
  "person": {
    "resourceId": "24912",
    "name": "Capture Worker"
  },
  "expiresAt": "2026-10-25T17:25:33.839Z",
  "actsAt": {
    "resourceId": "24911",
    "name": "Capture Station"
  },
  "viaSystem": "badge",
  "renewed": false
}
```

The same person again:

From `presence-create-renewed.json` — status 200:

```
{
  "presenceSessionId": "977",
  "person": {
    "resourceId": "24912",
    "name": "Capture Worker"
  },
  "expiresAt": "2026-10-25T17:25:33.866Z",
  "actsAt": {
    "resourceId": "24911",
    "name": "Capture Station"
  },
  "viaSystem": "badge",
  "renewed": true
}
```

Errors: 400 `system_required`, 400 `unknown_system`, 400 `alias_code_invalid`, 404 `alias_not_enrolled`, and the device errors above.

From `presence-not-enrolled.json` — status 404:

```
{
  "error": "alias_not_enrolled",
  "message": "No one is linked to that identifier."
}
```

GET /v1/presence-sessions/current

Who is checked in at this device now. Same fields as check-in, without `renewed`; 404 `no_active_session` when no one is. A session found past its `expiresAt` is ended then (recorded as `presence.checked_out`, reason `expired`) and answers 404.

From `presence-current.json` — status 200:

```
{
  "presenceSessionId": "977",
  "person": {
    "resourceId": "24912",
    "name": "Capture Worker"
  },
  "expiresAt": "2026-10-25T17:25:33.866Z",
  "actsAt": {
    "resourceId": "24911",
    "name": "Capture Station"
  },
  "viaSystem": "badge"
}
```

From `presence-current-none.json` — status 404:

```
{
  "error": "no_active_session",
  "message": "No one is checked in at this device."
}
```

POST /v1/presence-sessions/current/touch

Activity without a write: extends the current session from now and returns it, with `renewed: true`. Records nothing. 404 `no_active_session` when no one is checked in.

From `presence-touch.json` — status 200:

```
{
  "presenceSessionId": "977",
  "person": {
    "resourceId": "24912",
    "name": "Capture Worker"
  },
  "expiresAt": "2026-10-25T17:25:33.888Z",
  "actsAt": {
    "resourceId": "24911",
    "name": "Capture Station"
  },
  "viaSystem": "badge",
  "renewed": true
}
```

DELETE /v1/presence-sessions/current

Check the person out. Answers 204 with no body (`presence-end.json`) and records `presence.checked_out`, reason `tap_out`. 404 `no_active_session` when no one is checked in.

POST /v1/alias-enrollments/redeem

Link an identifier to a person, using a one-time enrolment code an administrator issued for that person in ReSync. Needs `acts_at_resource_id`. The enrolment must belong to the credential's organization.

Field	Type	Meaning
enrollmentCode	string	The 8-character code, with or without its hyphen.
code	string	The identifier as read; normalized by the enrolment's system's own rule.

Answers `{ person: { resourceId, name }, system }` and records `identity.alias_enrolled` on the person — the administrator who issued the code is its actor.

From `redeem-200.json` — status 200:

```
{
  "person": {
    "resourceId": "24913",
    "name": "Capture Linker"
  },
  "system": "badge"
}
```

Errors: 400 `enrollment_code_required`, 404 `enrollment_not_found` (wrong, used or cancelled), 410 `enrollment_expired`, 403 `enrollment_outside_owner_grant`, 400 `alias_code_invalid`, 409 `alias_already_bound`, and the device errors above.

From `redeem-outside-owner-grant.json` — status 403:

```
{
  "error": "enrollment_outside_owner_grant",
  "message": "That enrollment belongs to another organization than this device credential's."
}
```

Live updates

A device credential may open `GET /v1/live`, the server-sent event stream ReSync's web app uses, and follow the resources it can read: `PUT /v1/live/:connectionId/views` with `{ views: [{ resourceId, view }] }` (`view` one of `wall`, `members`, `inbox`, `canvas`, `lifecycle`; at most 50). The stream sends `hello { connectionId, heartbeatMs }`, then `changed { resourceId, view, parts }` whenever a followed view changes — read it again — and `resync` when you should re-read everything. A device credential's stream stays open until the key is revoked. Errors are under [Live updates](#).

Sign in with ReSync

How an app signs a person in is on its own page, [sign-in.md](#). The issuer's endpoints live at `https://auth.limnal.dev` (read them from discovery):

Endpoint	
GET /.well-known/openid-configuration	discovery
GET /.well-known/jwks.json	the signing keys (public members only)
POST /oauth/token	exchange an authorization code for tokens
GET /oauth/userinfo	who the signed-in person is

The two endpoints below serve ReSync's own consent screen. They take the person's ReSync session only — an app token answers `403 client_not_permitted`, a key `403 session_required`. Apps do not call them.

GET /v1/oauth/clients/:id

The consent-screen view of an **active** app: its name, its organization, and the NAMES of the types it may change (`operate`), attest on (`attest`) and see (`read`). With `?redirect_uri=`, `redirect_ok` says whether that URI is one the app registered (exact match; the list itself is never returned). A suspended, revoked or unknown app answers `404 connected_app_not_found`.

From `signin-consent.json` — status 200:

```
{
  "client_id": "01M5TZ7EPRR9VM4773MWF4EFVQ",
  "name": "Bid App",
  "owner_name": "Popular Design Mill & Cabinet, Inc.",
  "kind": "public",
  "allowed_types": {
    "operate": [
      "Bid Line"
    ],
    "attest": [],
    "read": [
      "Customer",
      "Bid",
      "Design Set",
      "Installation Line",
      "Adjustment",
      "Modification"
    ]
  },
  "status": "active",
  "redirect_ok": true
}
```

Errors: `400 invalid_client`, `404 connected_app_not_found`, `403 session_required`, `403 client_not_permitted`.

POST /v1/oauth/authorize

Mint a one-time authorization code for the signed-in person (it lives 60 seconds). Body: `client_id`, `redirect_uri` (exactly one the app registered), `scope` (`openid`, optionally `profile`), `state` (1–512 characters, echoed back), `code_challenge` (43–128 base64url characters), `code_challenge_method` (`S256`), `nonce` (optional, ≤ 255). The answer is `{ code, state }`; the consent screen then sends the browser to the redirect URI. This endpoint never redirects.

From `signin-authorize.json` — status 200:

```
{
  "code": "HHwsJDFbxkh2UUIXMxxsdUwc_vxwVz0V7QU5HnEkeIY",
  "state": "af0ifjsldkj"
}
```

Errors: 400 `invalid_client`, 404 `connected_app_not_found`, 400 `redirect_uri_mismatch`, 400 `unsupported_scope`, 400 `invalid_code_challenge`, 400 `invalid_state`, 400 `invalid_nonce`, 403 `session_required`, 403 `client_not_permitted`.

Resources — reading

GET /v1/resources

List resources of one type. **Needs read scope for the type.**

Parameter	Type	Default	Notes
<code>typeKey</code>	string	—	The type's code. Either this or <code>typeId</code> is required .
<code>typeId</code>	string	—	The type's numeric id, as an alternative to <code>typeKey</code> .
<code>statusId</code>	string	—	Restrict to one status.
<code>q</code>	string	—	Case-insensitive substring match on name and identifier.
<code>includeArchived</code>	boolean	false	Include archived resources.
<code>ownerResourceId</code>	string	—	Accepted only if it equals your owner; otherwise 403.
<code>limit</code>	string	50	Clamped to 1–100.
<code>cursor</code>	string	—	From a previous <code>nextCursor</code> .
<code>include</code>	string	—	Comma-separated. <code>properties</code> is the only member.
<code>tagId</code>	string	—	Only resources carrying this tag. Repeat it (<code>tagId=1&tagId=2</code>) to require every one.
<code>sort</code>	—	—	Not accepted on a key list: 400 <code>sort_not_supported</code> . A key list is always in <code>resourceId</code> order, so the cursor stays exact.

A parameter this door knows but cannot apply to a key (`sort`) is refused. A parameter it does not know at all is **ignored** — a misspelt `limit=5` returns the default page, not a 400 — so check spelling against this table. A signed-in person reading the same door gets `typeKey` and `include=properties` with the same meaning (an unknown `typeKey` is 404 `type_not_found` there too).

Results include resources of types **beneath** the one you name, restricted to types you may read.

Response: { `resources: [...]`, `nextCursor: string | null` }. Each row:

Field	Type	Meaning
<code>resourceId</code>	string	The id. A string — see conventions .
<code>name</code>	string	Display name.
<code>identifier</code>	string null	Your own identifier, if one was set.
<code>ownerResourceId</code>	number	Always your organization.
<code>resourceTypeId</code>	string	The type's id.
<code>resourceTypeName</code>	string	The type's display name — not stable, do not key on it.
<code>isSystem</code>	boolean	Whether the type is built in.
<code>createdAt</code>	string	ISO-8601 UTC.
<code>statusId</code>	number null	Current status, if the type has a lifecycle.
<code>statusName</code>	string null	Display name of that status.
<code>statusColor</code>	string null	Hex colour, for display.
<code>archivedAt</code>	string null	Set if archived.
<code>icon</code>	string	The type's icon, always one of <code>box</code> · <code>package</code> · <code>layers</code> · <code>hammer</code> · <code>wrench</code> · <code>ruler</code> · <code>truck</code> · <code>warehouse</code> · <code>factory</code> · <code>users</code> · <code>user</code> · <code>file-text</code> · <code>clipboard-list</code> · <code>tag</code> · <code>calendar</code> · <code>cog</code> · <code>circle-dollar-sign</code> · <code>door-open</code> (lucide icon names). The nearest icon the type or one of its ancestor types declares, counting only types your key can read; <code>box</code> when there is none.
<code>properties</code>	object	Only with <code>include=properties</code> .
<code>confirmedKeys</code>	object	Only with <code>include=properties</code> . Keys somebody has confirmed.

Errors: 400 type_required, 400 invalid_type_id, 400 invalid_status_id, 400 invalid_owner_resource_id, 400 invalid_cursor, 400 invalid_include, 400 invalid_tag_id, 400 sort_not_supported, 403 type_not_readable, 403 owner_grant_required, 403 read_types_required, 403 owner_forced_to_grant, 404 type_not_found.

From `list-operable-type.json` — status 200:

```
{
  "resources": [
    {
      "resourceId": "962",
      "name": "Slice-1 – Kitchen 3 · Bid",
      "identifier": null,
      "ownerResourceId": 5,
      "resourceTypeId": "176",
      "resourceTypeName": "Bid",
      "isSystem": false,
      "createdAt": "2026-08-02T17:41:46.784Z",
      "statusId": 282,
      "statusName": "Accepted",
      "statusColor": "#059669",
      "archivedAt": null,
      "icon": "box"
    },
    {
      "resourceId": "968",
      "name": "Slice-1 – Kitchen 3 · Bid 2",
      "identifier": null,
      "ownerResourceId": 5,
      "resourceTypeId": "176",
      "resourceTypeName": "Bid",
      "isSystem": false,
      "createdAt": "2026-08-02T18:00:58.209Z",
      "statusId": 282,
      "statusName": "Accepted",
      "statusColor": "#059669",
      "archivedAt": null,
      "icon": "box"
    }
  ],
  "nextCursor": "968"
}
```

With `include=properties`:

From `list-include-properties.json` — status 200:

```

{
  "resources": [
    {
      "resourceId": "13806",
      "name": "Dealer",
      "identifier": null,
      "ownerResourceId": 5,
      "resourceTypeId": "1264",
      "resourceTypeName": "Price Level",
      "isSystem": false,
      "createdAt": "2026-09-13T08:53:20.358Z",
      "statusId": null,
      "statusName": null,
      "statusColor": null,
      "archivedAt": null,
      "icon": "box",
      "properties": {
        "name": "Dealer",
        "pct_bp": 4536
      },
      "confirmedKeys": {}
    },
    {
      "resourceId": "13807",
      "name": "High-Volume Contractor",
      "identifier": null,
      "ownerResourceId": 5,
      "resourceTypeId": "1264",
      "resourceTypeName": "Price Level",
      "isSystem": false,
      "createdAt": "2026-09-13T08:53:22.255Z",
      "statusId": null,
      "statusName": null,
      "statusColor": null,
      "archivedAt": null,
      "icon": "box",
      "properties": {
        "name": "High-Volume Contractor",
        "pct_bp": 5025
      },
      "confirmedKeys": {}
    },
    {
      "resourceId": "13808",
      "name": "Contractor",
      "identifier": null,
      "ownerResourceId": 5,
      "resourceTypeId": "1264",
      "resourceTypeName": "Price Level",
      "isSystem": false,
      "createdAt": "2026-09-13T08:53:24.139Z",
      "statusId": null,
      "statusName": null,
      "statusColor": null,
      "archivedAt": null,
      "icon": "box",
      "properties": {
        "name": "Contractor",
        "pct_bp": 5678
      },
      "confirmedKeys": {}
    },
    {
      "resourceId": "13809",
      "name": "Homeowner",
      "identifier": null,

```

```

    "ownerResourceId": 5,
    "resourceTypeId": "1264",
    "resourceTypeName": "Price Level",
    "isSystem": false,
    "createdAt": "2026-09-13T08:53:26.009Z",
    "statusId": null,
    "statusName": null,
    "statusColor": null,
    "archivedAt": null,
    "icon": "box",
    "properties": {
      "name": "Homeowner",
      "pct_bp": 6937
    },
    "confirmedKeys": {}
  },
  {
    "resourceId": "13810",
    "name": "Internal",
    "identifier": null,
    "ownerResourceId": 5,
    "resourceTypeId": "1264",
    "resourceTypeName": "Price Level",
    "isSystem": false,
    "createdAt": "2026-09-13T08:53:27.881Z",
    "statusId": null,
    "statusName": null,
    "statusColor": null,
    "archivedAt": null,
    "icon": "box",
    "properties": {
      "name": "Internal",
      "pct_bp": 0
    },
    "confirmedKeys": {}
  }
],
"nextCursor": "13810"
}

```

An unknown `include` member is refused rather than ignored:

From `list-include-invalid.json` — status 400:

```

{
  "error": "invalid_include",
  "message": "unknown include member(s): foo (accepted: properties)"
}

```

GET /v1/resources/:id

One resource in full, including `properties`. **Needs read scope.**

It carries the identity fields a list row has (`resourceId`, `name`, `identifier`, `resourceTypeId`, `resourceTypeName`, `statusId`, `statusName`, `icon`, `createdAt`, `archivedAt`; the status colour is `statusColorCode` here), and beyond them `properties`, `confirmedKeys`, `propertySchema` (the shape the resource's type itself declares), `declaredPropertySchema` (that shape merged with every ancestor type's, a key declared nearer the resource's own type winning — every property the resource can carry), `derivedKeys` (properties computed by a formula — see [conventions](#)), `derivedFrom` / `derivatives` (provenance links), `edges` (connections made through a port on both ends) and `otherEdges` (the same row shape, for connections where either

end has no port — `thisResourcePortId` is this end's port or `null`; provenance links stay in `derivedFrom / derivatives`). Anything here your key cannot read follows the `names you cannot read` rule (`readable: false`). `readOnly` is `null`, or `{ statusName }` when the resource sits in a status its lifecycle declares read-only — its own fields, contents and outgoing connections then refuse writes (`409 status_read_only`).

Deprecated, informational only — will be removed in a future revision, with notice here first; do not branch on them: `isVerified` on resource reads (always false today) and `affectsInventory / affectsAvailability / affectsScheduling` on interaction-type reads (declared, never acted on).

Errors: `400 invalid_resource_id`, `403 subject_outside_grant`, `403 type_not_readable`, `403 owner_grant_required`, `404 resource_not_found` (nothing has that id — checked after your access, so a `404` never hides a resource you could read).

From `get-read-only-type.json` — status 200:

```
{
  "resourceId": "273",
  "resourceTypeId": "132",
  "ownerResourceId": "4",
  "name": "Contractor",
  "identifier": null,
  "properties": {
    "name": "Contractor",
    "pct_bp": 5678
  },
  "statusId": null,
  "isVerified": false,
  "createdAt": "2026-10-27T02:56:36.091Z",
  "archivedAt": null,
  "derivedFromTypeId": null,
  "resourceTypeName": "Price Level",
  "propertySchema": {
    "pct_bp": {
      "type": "number",
      "label": "List price multiplier (basis points)",
      "derivable": false
    }
  },
  "declaredPropertySchema": {
    "pct_bp": {
      "type": "number",
      "label": "List price multiplier (basis points)",
      "derivable": false
    }
  },
  "orgName": "Popular Design Mill & Cabinet, Inc.",
  "ownerOrgId": "3",
  "canvasMode": null,
  "statusName": null,
  "statusColorCode": null,
  "icon": "box",
  "lifecycleId": null,
  "canvasId": null,
  "derivedFrom": [],
  "derivatives": [],
  "derivedKeys": [],
  "edges": [],
  "otherEdges": [],
  "confirmedKeys": [],
  "readOnly": null
}
```

A resource owned by another organization:

From `get-outside-grant.json` — status 403:

```
{
  "error": "subject_outside_grant"
}
```

GET /v1/resources/:id/events

The resource's history, newest first. **Needs read scope.** See [events.md](#).

Parameter	Type	Default	Notes
limit	string	30	Clamped to 1–100. Note the default differs from the list.
cursor	string	—	From a previous <code>nextCursor</code> .
filter	string	all	<code>all</code> , <code>changes</code> , <code>messages</code> or <code>quiet</code> (see events.md). Anything else is treated as <code>all</code> .
after	string	—	Poll for changes: oldest first, from a stored <code>afterCursor</code> / <code>nextCursor</code> , or <code>start</code> . See events.md .

Response: `{ items: [...], nextCursor: string | null, afterCursor: string }`. With `after`, the order is oldest first and `nextCursor` is always present: it is where your next poll starts.

Errors: `400 invalid_resource_id`, `400 invalid_cursor` (an `after` this API did not issue), `403 subject_outside_grant`, `403 type_not_readable`, `403 owner_grant_required`.

From `events-outside-grant.json` — status 403:

```
{
  "error": "subject_outside_grant"
}
```

GET /v1/resources/:id/members

Resources currently contained by this one. **Needs read scope.** Returns an array, one row per member: `childResourceId`, `childResourceTypeId`, `childName`, `childTypeName`, `componentStatus`, `createdAt`, `roles`, `enrollmentCount` and `applicableActions`.

From `members-read.json` — status 200:

```
[
  {
    "childResourceId": 24908,
    "childResourceTypeId": 3425,
    "childName": "Capture line",
    "childTypeName": "Bid Line",
    "componentStatus": "active",
    "createdAt": "2026-10-21T07:57:07.796Z",
    "roles": [],
    "enrollmentCount": 1,
    "applicableActions": []
  },
  {
    "childResourceId": 24911,
    "childResourceTypeId": 3425,
    "childName": "Capture line two",
    "childTypeName": "Bid Line",
    "componentStatus": "active",
    "createdAt": "2026-10-21T07:57:07.848Z",
    "roles": [],
    "enrollmentCount": 1,
    "applicableActions": []
  }
]
```

GET /v1/resources/:id/containers

The containers this resource is currently inside — the other direction from `/members`. **Needs read scope** on the resource. A container your key cannot read is left out, as if it were not there.

Response: { `containers`: [...] }. Each row: `parentResourceId`, `name`, `resourceTypeId`, `resourceTypeName`, `statusName`, `statusColorCode`, `archivedAt`, `membershipCount` (how many active memberships this resource holds in that container) and `outboundFeeds` (the destinations that container declares for moving what it holds on — only ones your key can read and a move would accept).

From `containers-read.json` — status 200:

```
{
  "containers": [
    {
      "parentResourceId": "24909",
      "name": "Capture set A",
      "resourceTypeId": "1171",
      "resourceTypeName": "Design Set",
      "statusName": "Draft",
      "statusColorCode": null,
      "archivedAt": null,
      "membershipCount": 1,
      "outboundFeeds": []
    }
  ]
}
```

Errors: 400 `invalid_resource_id`, 403 `subject_outside_grant`, 403 `type_not_readable`, 404 `resource_not_found`.

From `containers-outside-grant.json` — status 403:

```
{
  "error": "subject_outside_grant"
}
```

GET /v1/resources/search

Find resources by name across types — the read behind a picker. **Needs read scope:** only resources your key can read are returned.

Parameter	Type	Notes
q	string	Required in practice. Case-insensitive substring of the name. Fewer than 2 characters returns <code>[]</code> .
types	string	Comma-separated type codes; types beneath each are included. <code>all</code> searches every type. Defaults to <code>person,organization</code> . A key never searches people: <code>person</code> is dropped without notice.
enrollInto	string	A container id. Leaves out exactly what <code>POST /v1/resources/:id/members</code> on that container would refuse (<code>containment_cycle</code> , <code>containment_cap</code>). Needs operate scope on the container, as the act does.
moveOf	string	A resource you are about to move. Leaves out the destinations a move would refuse: the resource itself, anything inside it, and containers whose cap it has reached. Needs operate scope on that resource.
moveFrom	string	With <code>moveOf</code> : the container it leaves, so the cap check is exact. Needs operate scope on it.
restrictToOrgId	string	A workspace filter for signed-in people. Ignored for a key — a key always searches its own readable set.

Response: a bare array (no envelope), at most 20 rows in all — however many types you ask for — ordered by name. Only ordinary resources are returned; the records ReSync keeps to describe a type or a rule never appear. Each row: `resourceId`, `resourceTypeId`, `resourceTypeCode`, `resourceTypeName`, `name`, `ownerResourceId`, `icon`.

Here only one of three matching lines may be added to the set: the other two are already inside a set, and a line's type allows one.

From `search-enroll-into.json` — status 200:

```
[
  {
    "resourceId": "24912",
    "resourceTypeId": "3425",
    "resourceTypeCode": "cab_line",
    "resourceTypeName": "Bid Line",
    "name": "Capture line loose",
    "ownerResourceId": "5",
    "icon": "box"
  }
]
```

Errors: 400 `invalid_enroll_into`, 400 `invalid_move_of`, 400 `invalid_move_from`, 403 `subject_outside_grant`, 403 `type_not_operable`, 403 `operate_types_required`.

From `search-enroll-into-outside-grant.json` — status 403:

```
{
  "error": "subject_outside_grant"
}
```

GET /v1/resources/:id/advancement

One call returning the resource, the status moves available to it, and the requirements for each. **Needs read scope.**
Response: { resource, transitions, requirements }.

Note — not available to keys. GET /v1/resources/:id/why (a resource's status, key facts — each with its display label and value — recent status history, each available move with one plain-language sentence, and the explanation lines the ReSync app shows) is served to signed-in people only. A key gets 401 there. Build the same picture from /advancement and /events.

A sent bid — its three ways on (Accept, Decline, Supersede) and the checklist for each:

From `bid-advancement.json` — status 200:

```

{
  "resource": {
    "resourceId": "4704",
    "resourceTypeId": "27",
    "ownerResourceId": "4",
    "name": "Harbor Homes – kitchen",
    "identifier": null,
    "properties": {
      "tax_rate_bp": 825
    },
    "statusId": "25",
    "isVerified": false,
    "createdAt": "2026-10-27T04:13:17.988Z",
    "archivedAt": null,
    "derivedFromTypeId": null,
    "resourceTypeName": "Bid",
    "propertySchema": {
      "tax_rate_bp": {
        "type": "number",
        "label": "Tax rate (basis points)",
        "derivable": false
      }
    },
    "declaredPropertySchema": {
      "tax_rate_bp": {
        "type": "number",
        "label": "Tax rate (basis points)",
        "derivable": false
      }
    },
    "orgName": "Popular Design Mill & Cabinet, Inc.",
    "ownerOrgId": "3",
    "canvasMode": null,
    "statusName": "Sent",
    "statusColorCode": null,
    "icon": "box",
    "lifecycleId": 6,
    "canvasId": null,
    "derivedFrom": [],
    "derivatives": [],
    "derivedKeys": [],
    "edges": [
      {
        "interactionId": 839,
        "typeName": "bid_for",
        "direction": "out",
        "otherResourceId": 4703,
        "otherResourceName": "Harbor Homes",
        "thisResourcePortId": 4143
      },
      {
        "interactionId": 840,
        "typeName": "priced_at",
        "direction": "out",
        "otherResourceId": 273,
        "otherResourceName": "Contractor",
        "thisResourcePortId": 4144
      },
      {
        "interactionId": 841,
        "typeName": "pays_on",
        "direction": "out",
        "otherResourceId": 278,
        "otherResourceName": "Net 30",
        "thisResourcePortId": 4145
      }
    ]
  }
}

```

```

    ],
    "otherEdges": [],
    "confirmedKeys": [],
    "readOnly": {
      "statusName": "Sent"
    }
  },
  "transitions": [
    {
      "transitionId": 30,
      "fromStatusId": 25,
      "toStatusId": 27,
      "actionLabel": "Decline",
      "autoAdvance": false,
      "toStatusName": "Declined",
      "toStatusPreventsUsage": false,
      "toStatusIsAllocating": false,
      "toStatusSubjectReadOnly": true,
      "records": null
    },
    {
      "transitionId": 31,
      "fromStatusId": 25,
      "toStatusId": 28,
      "actionLabel": "Supersede",
      "autoAdvance": false,
      "toStatusName": "Superseded",
      "toStatusPreventsUsage": false,
      "toStatusIsAllocating": false,
      "toStatusSubjectReadOnly": true,
      "records": null
    },
    {
      "transitionId": 29,
      "fromStatusId": 25,
      "toStatusId": 26,
      "actionLabel": "Accept",
      "autoAdvance": true,
      "toStatusName": "Accepted",
      "toStatusPreventsUsage": false,
      "toStatusIsAllocating": false,
      "toStatusSubjectReadOnly": true,
      "records": null
    }
  ],
  "requirements": {
    "29": [
      {
        "requirementId": 39,
        "name": "Bid accepted (emission-gated)",
        "blocking": "hard",
        "satisfied": false,
        "missing": [
          "Accept bid"
        ],
        "leaves": [
          {
            "leaf": "evidence",
            "label": "Accept bid",
            "caption": "not recorded yet",
            "satisfied": false,
            "attestable": true,
            "actionLabel": "Accept bid",
            "eventType": "bid.accepted",
            "payloadKey": null,
            "payloadPredicate": null,

```

```

        "actor": null,
        "actorRoles": [],
        "payloadFields": [],
        "repeatable": false,
        "blockedByActor": false,
        "satisfiedEver": false,
        "mayEmitUnvocab": false
      }
    ]
  },
  "30": [],
  "31": []
}

```

Each move also carries `records`: the event the move records on the resource when it completes, as `{ eventType, label }`, or `null` when it records nothing. On a sent bid all three are `null`. On an accepted bid, the move to Converted reads `"records": { "eventType": "bid.converted", "label": "Convert to order" }` (`bid-advancement-accepted.json`): moving the bid records the conversion, so you do not post it yourself (see `PATCH /v1/resources/:id/status`).

GET /v1/resources/:id/capabilities

The actions declared on this resource. **Needs read scope.** Response: `{ capabilities: [...] }` — empty when the resource's type declares none, as for a bid:

From `bid-capabilities.json` — status 200:

```

{
  "capabilities": []
}

```

GET /v1/resources/:id/requirements

The requirement checklist for one status move. **Needs read scope.**

Parameter	Type	Notes
transitionId	string	Required. From <code>/advancement</code> or <code>/state-transitions</code> .

Response: `{ requirements: [...] }` — the same array a blocked status change returns, so one shape to learn (the blocked status change is shown under `PATCH /v1/resources/:id/status`). A design set's Freeze checklist:

From `requirements-read.json` — status 200:

```

{
  "requirements": [
    {
      "requirementId": 43,
      "name": "R-SET-FREEZE – all lines frozen",
      "blocking": "hard",
      "satisfied": false,
      "missing": [
        "All lines frozen"
      ],
      "leaves": [
        {
          "leaf": "members",
          "label": "All lines frozen",
          "caption": "0/5 members at required status",
          "satisfied": false,
          "attestable": false,
          "actionLabel": null,
          "actor": null,
          "actorRoles": [],
          "payloadFields": [],
          "blockedByActor": false,
          "satisfiedEver": false,
          "mayEmitUnvocab": false,
          "memberTotal": 5,
          "memberSatisfied": 0
        }
      ]
    },
    {
      "requirementId": 44,
      "name": "Contents compatible",
      "blocking": "hard",
      "satisfied": true,
      "missing": [],
      "leaves": [
        {
          "leaf": "subtree_compat",
          "label": "No design set settings conflicts",
          "caption": "everything inside fits",
          "satisfied": true,
          "attestable": false,
          "actionLabel": null,
          "actor": null,
          "actorRoles": [],
          "payloadFields": [],
          "blockedByActor": false,
          "satisfiedEver": false,
          "mayEmitUnvocab": false,
          "violations": [],
          "violationCount": 0,
          "subtreeCompatVacuous": false
        }
      ]
    }
  ]
}

```

Each requirement carries `leaves[]`, one entry per condition. Every leaf has a readable `label` (what the condition is) and a `caption` (where it stands for this resource, e.g. "missing", "present", "not recorded yet"). `missing[]` lists the labels of the unmet ones. A leaf with `leaf: "subtree_compat"` checks that everything inside the resource fits the settings passed down to it. It additionally carries `violationCount`, `violations` (the first 10: `descendantResourceId`, `ancestorResourceId`, `kindTypeId`, `denyEdgeId`, `layer`, `code` — `compat_deny`

or `contents_locked` — and a readable `message`) and `subtreeCompatVacuous` (true when nothing inside receives a setting, so the check passes by default). An `id` is left out when your key cannot read that resource. The same fields appear in `/advancement` and in the `422` body of a blocked `PATCH /v1/resources/:id/status` .

Errors: `400 transitionId is required` , `400 Invalid transitionId` , `403 as above` , `404 not found` .

Resources — writing

`PATCH /v1/resources/:id`

Change a resource. **Needs operate scope**.

Field	Type	Notes
<code>name</code>	string	Non-empty. Ignored if the type computes its name from properties.
<code>identifier</code>	string null	Your own identifier. Trimmed; empty becomes <code>null</code> .
<code>properties</code>	object	Merged into the existing map, not replaced.
<code>actorResourceId</code>	number	Optional. Record a specific person as the actor. Must be a person your key can read; otherwise <code>403 actor_not_readable</code> .
<code>actorAliases</code>	object	Optional. <code>{ system, code }</code> — record an external identity as the actor. Resolved under your organization; an identity seen for the first time is created there. The two actor fields are mutually exclusive.
<code>presenceSessionId</code>	number	Device credentials with <code>actor_source</code> only — the <code>id</code> from check-in; a string or a number is accepted. Any other key leaves it out.

The same three actor fields are accepted, with the same rules, by every key write that records an act: `POST /v1/resources/:id/members` , `POST /v1/resources/:id/exit` , `POST /v1/resources/:id/move` , `POST /v1/resources/:id/attestations` , `PATCH /v1/resources/:id/status` and `POST /v1/resources/find-or-create` . A malformed actor is refused with an English sentence in `error` (for example `"actorResourceId and actorAlias are mutually exclusive"`), not a code — see [errors.md](#).

Omit both actor fields and the write is attributed to your key's own service identity, which is the usual case. See [events.md](#).

Errors: `400 invalid_resource_id` , `400 invalid_resource_update` (a blank `name`), `400 derive_target_write_blocked` (with keys), `409 confirmed_key_locked` (with keys), `409 status_read_only` (the resource sits in a read-only status, for example a sent bid), `409 unique_key_taken` (another resource of your organization already holds that value of a declared unique key), `403 subject_outside_grant` , `403 type_not_operable` , `403 operate_types_required` , `403 owner_grant_required` , `403 actor_not_readable` , `404 resource_not_found` .

A resource in a read-only status:

From `status-read-only-409.json` — `status 409`:

```
{
  "error": "status_read_only",
  "message": "Harbor Homes – kitchen is Sent, a read-only status – its properties cannot be changed.",
  "resourceId": 24911,
  "statusId": 1314
}
```

A value a declared unique key already holds — here a modification given the identifier of one in the price book:

From `unique-key-taken-409.json` — status 409:

```
{
  "error": "unique_key_taken",
  "message": "NO BOTTOM is already used by another modification owned by Popular Design Mill & Cabinet, Inc.",
  "holderResourceId": "8890",
  "key": "properties.sku",
  "value": "NO BOTTOM"
}
```

From `patch-producer-id.json` — status 200:

```
{
  "resourceId": "14211",
  "resourceTypeId": "176",
  "ownerResourceId": "5",
  "name": "MSDEV Terms Verify (msdev:terms-verify-pos)",
  "identifier": null,
  "properties": {
    "name": "MSDEV Terms Verify (msdev:terms-verify-pos)",
    "recon_beat149": "p2"
  },
  "statusId": "280",
  "isVerified": false,
  "confirmedKeys": {},
  "createdAt": "2026-09-14T02:00:12.772Z"
}
```

A type you may read but not operate:

From `patch-read-only-type.json` — status 403:

```
{
  "error": "type_not_operable"
}
```

A property somebody has confirmed — note that the **whole** request is refused:

From `patch-confirmed-locked.json` — status 409:

```
{
  "error": "confirmed_key_locked",
  "message": "confirmed_key_locked: 'depth' is confirmed – a confirmed value is a human's assertion and cannot be overwritten by a key",
  "code": "confirmed_key_locked",
  "keys": [
    "depth"
  ]
}
```

POST /v1/resources/:id/members

Put a resource inside this one. **Needs operate scope on both.**

Field	Type	Notes
childResourceId	number	Required. Positive integer.
componentRoleId	number	Optional. Must belong to the container's type.
actorResourceId / actorAlias		Optional, as for PATCH.

Returns 201 with the membership row:

From `members-enrol.json` — status 201:

```
{
  "componentId": 262526,
  "parentResourceId": 24911,
  "childResourceId": 24912,
  "componentRole": null,
  "sourceSlotId": null,
  "sourceRuleId": null,
  "componentStatus": "active",
  "statusChangedAt": "2026-10-21T18:47:22.424Z",
  "sortOrder": 0,
  "properties": {},
  "createdAt": "2026-10-21T18:47:22.424Z"
}
```

Errors: 400 child_resource_id_required, 400 invalid_component_role_id, 400 self_membership, 400 role_type_mismatch, 403 subject_outside_grant, 403 type_not_operable, 403 operate_types_required, 403 actor_not_readable, 404 role_not_found, 404 resource_not_found, 409 already_a_member, 409 status_read_only (the container sits in a read-only status), 422 compat_deny, 422 contents_locked, 422 containment_cycle, 422 containment_cap.

A container can pass a setting down to what it contains (for example a material chosen once for a whole set). Putting a resource inside such a container checks that resource, and everything inside it, against the settings it would now receive; a conflict is refused with 422 and nothing is written.

POST /v1/resources/:id/exit

Take this resource out of a container. **Needs operate scope on both.** The membership is marked exited, never deleted — the history stays.

Field	Type	Notes
containerId	number	Required. The container to leave.
actorResourceId / actorAlias		Optional, as for <code>PATCH</code> .

Response: { exited: true, componentId: number }.

From `exit.json` — status 200:

```
{
  "exited": true,
  "componentId": 261680
}
```

Errors: 400 invalid_resource_id, 400 container_id_required, 403 subject_outside_grant, 403 type_not_operable, 403 operate_types_required, 403 actor_not_readable, 404 resource_not_found, 409 not_an_active_member, 409 status_read_only (the container sits in a read-only status).

POST /v1/resources/:id/move

Move this resource from one container to another in one step. **Needs operate scope on the resource and on both containers.**

Field	Type	Notes
fromContainerId	number	Required. The container it is in now.
toContainerId	number	Required. The container it goes to. Must differ from <code>fromContainerId</code> .
actorResourceId / actorAlias		Optional, as for <code>PATCH</code> .

Both halves happen in one transaction: the old membership is exited (`component.exited`, carrying `moved_to_resource_id`) and the new one enrolled (`component.enrolled`, carrying `moved_from_resource_id`). Either both are recorded or neither is. The destination is checked exactly as an enrolment is (`compat_deny`, `contents_locked`, `containment_cycle`, `containment_cap`); a containment cap does not count the container being left. `GET /v1/resources/search?moveOf=` lists the destinations this door would accept.

Response: { moved: true, exitedComponentId, enrolledComponentId }.

From `move.json` — status 200:

```
{
  "moved": true,
  "exitedComponentId": 261679,
  "enrolledComponentId": 261681
}
```

Errors: 400 invalid_resource_id, 400 container_id_required, 403 subject_outside_grant, 403 type_not_operable, 403 operate_types_required, 404 resource_not_found, 409 already_a_member (the two containers are the same, or it is already in the destination), 409 not_an_active_member (it is not in

fromContainerId), 409 status_read_only (either container sits in a read-only status), 403 actor_not_readable, 422 compat_deny, 422 contents_locked, 422 containment_cycle, 422 containment_cap.

From `move-same-container.json` — status 409:

```
{
  "error": "already_a_member",
  "message": "fromContainerId and toContainerId must differ"
}
```

POST /v1/resources/:id/attestations

Record that something happened. **Needs attest or operate scope.** This is how a fact enters the ledger without changing the resource's own fields — *inspected*, *sent*, *received*.

Field	Type	Notes
eventType	string	Required. Must be a declared type — see <code>GET /v1/attestation-event-types</code> .
payload	object	Flat object of string/number/boolean values. No nesting, no arrays.
scopeResourceId	number	Optional. Record the event against a container as well. It must be a resource your key can read; otherwise 403 subject_outside_grant and nothing is recorded.
actorResourceId / actorAlias		Optional, as for <code>PATCH</code> .

Returns 201 { eventUlid, eventType, resourceId, actorResourceId } — actorResourceId is your key's service identity unless the body named a person:

From `bid-attest-sent.json` — status 201:

```
{
  "eventUlid": "01M5FWMHJY4Q8TB2KSHCS8Z9S9",
  "eventType": "bid.sent",
  "resourceId": 24911,
  "actorResourceId": 24907
}
```

One per status, unless declared repeatable. An event type is either *repeatable* (repeatable: true on `GET /v1/attestation-event-types`) or recorded once per resource for as long as the resource stays in its current status. A second one in the same status answers 409 attestation_not_repeatable; after the resource moves to another status (or back), it can be recorded again.

Errors: 400 invalid_resource_id; 403 subject_outside_grant, 403 type_not_operable, 403 operate_types_required, 403 actor_not_readable; 404 resource not found; 409 attestation_not_repeatable; 422 attestation_status_gate_unmet when the type requires the resource to be at a particular status first (requiredSubjectStatusId on the type). Both checks happen atomically with the write, and a refused attestation records nothing.

From `attestation-409-not-repeatable.json` — status 409:

```
{
  "error": "attestation_not_repeatable",
  "message": "Send bid is already recorded for Harbor Homes – kitchen in its current status.",
  "resourceId": 24911,
  "eventType": "bid.sent"
}
```

From `attestation-422-gate-unmet.json` — status 422:

```
{
  "error": "attestation_status_gate_unmet",
  "requiredSubjectStatusId": 280,
  "currentStatusId": 1314
}
```

The status check runs first: a non-repeatable type sent again after the resource has moved on answers `422`, not `409`. Its body-validation refusals (unknown `eventType`, non-flat `payload`, missing declared fields) return a sentence in `error` rather than a code — see [errors.md](#).

From `attestation-scope-outside-grant.json` — status 403:

```
{
  "error": "subject_outside_grant"
}
```

PATCH /v1/resources/:id/status

Move a resource to a new status. **Needs attest or operate scope** — a status move is an *attest* act (like attestations).

Field	Type	Notes
<code>toStatusId</code>	number	Required. Must be a legal move from the current status (<code>GET /v1/state-transitions</code>).
<code>actorResourceId</code> / <code>actorAlias</code>		Optional, as for <code>PATCH</code> .

Returns `200 { resourceId, fromStatusId, statusId }` (plus `advisories` when a soft requirement is unmet):

From `patch-status-200.json` — status 200:

```
{
  "resourceId": 24913,
  "fromStatusId": 1318,
  "statusId": 1319
}
```

422 when a hard requirement is unmet. The body is the same `requirements` array as `GET /v1/resources/:id/requirements` — every requirement on the move, met or not — under a sentence in `error` :

From `patch-status-422-requirements.json` — status 422:

```

{
  "error": "Transition blocked by unmet requirements",
  "requirements": [
    {
      "requirementId": 43,
      "name": "R-SET-FREEZE – all lines frozen",
      "blocking": "hard",
      "satisfied": false,
      "missing": [
        "All lines frozen"
      ],
      "leaves": [
        {
          "leaf": "members",
          "label": "All lines frozen",
          "caption": "0/5 members at required status",
          "satisfied": false,
          "attestable": false,
          "actionLabel": null,
          "actor": null,
          "actorRoles": [],
          "payloadFields": [],
          "blockedByActor": false,
          "satisfiedEver": false,
          "mayEmitUnvocab": false,
          "memberTotal": 5,
          "memberSatisfied": 0
        }
      ]
    },
    {
      "requirementId": 44,
      "name": "Contents compatible",
      "blocking": "hard",
      "satisfied": true,
      "missing": [],
      "leaves": [
        {
          "leaf": "subtree_compat",
          "label": "No design set settings conflicts",
          "caption": "everything inside fits",
          "satisfied": true,
          "attestable": false,
          "actionLabel": null,
          "actor": null,
          "actorRoles": [],
          "payloadFields": [],
          "blockedByActor": false,
          "satisfiedEver": false,
          "mayEmitUnvocab": false,
          "violations": [],
          "violationCount": 0,
          "subtreeCompatVacuous": false
        }
      ]
    }
  ]
}

```

A move the lifecycle does not declare — here a frozen line back to Draft:

From `patch-status-409-invalid-transition.json` — status 409:

```
{
  "error": "Invalid status transition"
}
```

Moves that record an event. A move may record an event as it completes — its `records` in `GET /v1/resources/:id/advancement` names it. The event is appended on the resource in the same step as the status change, with you as its actor, and anything that follows from it (the order a converted bid produces) follows from the move. Do **not** post that event yourself: once the move has recorded it, a `POST .../attestations` of the same event returns `409 attestation_not_repeatable`:

From `bid-attest-converted-409.json` — status 409:

```
{
  "error": "attestation_not_repeatable",
  "message": "Convert to order is already recorded for Harbor Homes – kitchen in its current status.",
  "resourceId": 24911,
  "eventType": "bid.converted"
}
```

If the event cannot be recorded, **nothing moves** — the status, its history row and the event are all left as they were — and you get the attestation door's own refusal: `422 attestation_status_gate_unmet` or `409 attestation_not_repeatable`, or `409 transition_record_undeclared` when the event the move names is not declared for this resource (a configuration problem — see [errors](#)).

Errors: `403 subject_outside_grant`, `403 type_not_operable`, `403 operate_types_required`, `403 actor_not_readable`; `404 resource not found`; `409 Resource is already in this status`; `409 Invalid status transition`; `422 requirements unmet`; the three refusals above, on a move that records an event. Like attestations, this door returns sentences in `error` rather than codes for its own refusals (the three refusals of a recorded event are codes).

If another request moves the resource to the same status at the same moment, both succeed: the one that arrives second returns `200` with the same body and records nothing, so the status change is recorded once. If the other request moved it somewhere else, you get `409 Invalid status transition`.

Ports and attachments

A resource connects to other resources through **ports**. A port that allows one connection and accepts one type is a **field**: it holds a single choice, for example the material a set is made of. The connection made through a field is an **attachment**. A field may also **pass its choice down** to the resources the holder contains (`contentsScope` : `none` , `inherit` — a contained resource may choose its own — or `locked`).

Attaching is checked against the compatibility rules: the new value against the holder, against what the holder is already connected to, against the choices the holder inherits, and — when the field passes down — against everything inside the holder that would receive it. A refusal names what conflicts in `message`.

Who may connect. You need **write** on the side whose port points **out** (the side making the choice) and only **read** on the side whose port points **in** (the shared value being chosen). Compatibility markings themselves (`compatible_with` / `incompatible_with`) are authored by signed-in people only.

GET `/v1/resources/:id/ports`

The resource's active ports. **Needs read scope.** Each port carries `portName`, `direction`, `types` (accepted types set on this port), `effectiveTypes` (what it actually accepts), `maxConnections`, `defaultInteractionTypeId` and `contentsScope` (from its definition; `null` for a port with none).

Every port also carries:

- **isField** — `true` on a **field**: a port that points out, takes one connection and accepts exactly one type (the kind of value it holds). `false` on every other port. Use it to decide when to present a port as a single-choice value rather than a list of connections.
- **allowedInteractionTypes** — `[{ id, name }]`, sorted by id: the interaction types a connection on this port may use. Offer exactly this list when you let someone pick a connection type; a type outside it is refused when you connect. Compatibility types (`compatible_with` / `incompatible_with`) are never in it.
- **setting** — on a **field** (a port that points out, takes one connection and accepts exactly one type): the value the field resolves to and where it comes from. `null` on every other port, and on a field where nothing resolves. Shape:

Field	Meaning
value	{ resourceId, name } — the value to show for the field.
scope	own (set on this resource) or inherited (set on an enclosing item and passed down).
locked	true when an enclosing item locks this setting for everything inside it.
source	{ resourceId, name } of the enclosing item the value or lock comes from; null for an own value with no lock above it.
overrides	{ value, source } — the inherited value an own value replaces, or null.
conflicts	[{ value, source }] — when enclosing items pass down different values, every one of them; when an own value differs from a lock above it, the lock. Otherwise [].
note	One line of display text describing the above, or null. Show it as-is; do not build your own.
clear	On a field holding its own value: { label, edgeId }. label is display text naming what applies once the own value is removed (show it as-is); delete the connection edgeId (DELETE /v1/resource-interactions/:id) to fall back to it. null when the value is inherited.
deviates	true when an own value differs from the inherited value it replaces (a deviation from what the enclosing item passes down); false otherwise. An own value that differs from a lock above it is a conflict (conflicts), never a deviation.

A resource your key cannot read is never named: its `resourceId` comes back `null`, its `name` is the word "a setting" (for a value) or "an enclosing item" (for a source), and the object carries `readable: false`; `note` uses the same words.

- **typeChoices** — [{ `resourceTypeId`, `name`, `icon` }], sorted by name: the types that may still be **added** to this port's accepted types — readable to your key, accepted by the port's definition (a type it declares, or a subtype of one; any readable type when the port has no definition or its definition declares none), and not already on the port. Offer exactly this list; signed-in people adding a type outside it are refused. `icon` is the type's icon (see `GET /v1/resources`).

Optional `?workspace=<ownerResourceId>` narrows `typeChoices` to global types and that owner's types. An owner outside your key's reach is `400 workspace_not_in_scope`; a value that is not an id is `400 invalid_workspace`.

From `ports-read.json` — status 200:

```
[
  {
    "resourcePortId": 1485,
    "resourceId": 13811,
    "portName": "Species",
    "direction": "out",
    "description": null,
    "isActive": true,
    "createdAt": "2026-09-20T09:44:59.109Z",
    "scope": "boundary",
    "sortOrder": 0,
    "routingRule": {},
    "required": false,
    "expectedRoleId": null,
    "derivedFromPortDefinitionId": 478,
    "types": [
      {
        "resourcePortTypeId": 396,
        "resourcePortId": 1485,
        "resourceTypeId": 1404,
        "typeName": "Species",
        "parameters": {}
      }
    ],
    "maxConnections": 1,
    "defaultInteractionTypeId": 135,
    "contentsScope": "inherit",
    "effectiveTypes": [
      {
        "resourceTypeId": 1404,
        "typeName": "Species"
      }
    ],
    "setting": {
      "value": {
        "resourceId": 7267,
        "name": "Laminate"
      },
      "scope": "own",
      "clear": {
        "edgeId": 1601,
        "label": "Clear"
      },
      "deviates": false,
      "locked": false,
      "source": null,
      "overrides": null,
      "conflicts": [],
      "note": null
    },
    "allowedInteractionTypes": [
      {
        "id": 1,
        "name": "allocates"
      },
      {
        "id": 2,
        "name": "consumes"
      },
      {
        "id": 3,
        "name": "reserves"
      },
      {
        "id": 4,
        "name": "produces"
      }
    ]
  }
]
```

```

    },
    {
      "id": 5,
      "name": "depends_on"
    },
    {
      "id": 6,
      "name": "assigned_to"
    },
    {
      "id": 19,
      "name": "certified_in"
    },
    {
      "id": 20,
      "name": "licensed_for"
    },
    {
      "id": 21,
      "name": "derives_from"
    },
    {
      "id": 22,
      "name": "feeds"
    },
    {
      "id": 23,
      "name": "bid_for"
    },
    {
      "id": 24,
      "name": "priced_at"
    },
    {
      "id": 25,
      "name": "pays_on"
    },
    {
      "id": 40,
      "name": "mounts_on"
    },
    {
      "id": 135,
      "name": "uses"
    },
    {
      "id": 278,
      "name": "sells"
    }
  ],
  "isField": true,
  "typeChoices": []
},
{
  "resourcePortId": 1503,
  "resourceId": 13811,
  "portName": "Front",
  "direction": "out",
  "description": null,
  "isActive": true,
  "createdAt": "2026-09-20T10:07:33.257Z",
  "scope": "boundary",
  "sortOrder": 0,
  "routingRule": {},
  "required": false,
  "expectedRoleId": null,

```

```

"derivedFromPortDefinitionId": 480,
"types": [
  {
    "resourcePortTypeId": 414,
    "resourcePortId": 1503,
    "resourceTypeId": 1406,
    "typeName": "Front",
    "parameters": {}
  }
],
"maxConnections": 1,
"defaultInteractionTypeId": 135,
"contentsScope": "inherit",
"effectiveTypes": [
  {
    "resourceTypeId": 1406,
    "typeName": "Front"
  }
],
"setting": {
  "value": {
    "resourceId": 7369,
    "name": "Modern Horizon"
  },
  "scope": "own",
  "clear": {
    "edgeId": 2148,
    "label": "Clear"
  },
  "deviates": false,
  "locked": false,
  "source": null,
  "overrides": null,
  "conflicts": [],
  "note": null
},
"allowedInteractionTypes": [
  {
    "id": 1,
    "name": "allocates"
  },
  {
    "id": 2,
    "name": "consumes"
  },
  {
    "id": 3,
    "name": "reserves"
  },
  {
    "id": 4,
    "name": "produces"
  },
  {
    "id": 5,
    "name": "depends_on"
  },
  {
    "id": 6,
    "name": "assigned_to"
  },
  {
    "id": 19,
    "name": "certified_in"
  },
  {

```

```

        "id": 20,
        "name": "licensed_for"
    },
    {
        "id": 21,
        "name": "derives_from"
    },
    {
        "id": 22,
        "name": "feeds"
    },
    {
        "id": 23,
        "name": "bid_for"
    },
    {
        "id": 24,
        "name": "priced_at"
    },
    {
        "id": 25,
        "name": "pays_on"
    },
    {
        "id": 40,
        "name": "mounts_on"
    },
    {
        "id": 135,
        "name": "uses"
    },
    {
        "id": 278,
        "name": "sells"
    }
],
"isField": true,
"typeChoices": []
},
{
    "resourcePortId": 2583,
    "resourceId": 13811,
    "portName": "Finish",
    "direction": "out",
    "description": null,
    "isActive": true,
    "createdAt": "2026-10-17T05:33:22.749Z",
    "scope": "boundary",
    "sortOrder": 0,
    "routingRule": {},
    "required": false,
    "expectedRoleId": null,
    "derivedFromPortDefinitionId": 854,
    "types": [
        {
            "resourcePortTypeId": 784,
            "resourcePortId": 2583,
            "resourceTypeId": 1405,
            "typeName": "Finish",
            "parameters": {}
        }
    ],
    "maxConnections": 1,
    "defaultInteractionTypeId": 135,
    "contentsScope": "inherit",
    "effectiveTypes": [

```

```

    {
      "resourceTypeId": 1405,
      "typeName": "Finish"
    }
  ],
  "setting": null,
  "allowedInteractionTypes": [
    {
      "id": 1,
      "name": "allocates"
    },
    {
      "id": 2,
      "name": "consumes"
    },
    {
      "id": 3,
      "name": "reserves"
    },
    {
      "id": 4,
      "name": "produces"
    },
    {
      "id": 5,
      "name": "depends_on"
    },
    {
      "id": 6,
      "name": "assigned_to"
    },
    {
      "id": 19,
      "name": "certified_in"
    },
    {
      "id": 20,
      "name": "licensed_for"
    },
    {
      "id": 21,
      "name": "derives_from"
    },
    {
      "id": 22,
      "name": "feeds"
    },
    {
      "id": 23,
      "name": "bid_for"
    },
    {
      "id": 24,
      "name": "priced_at"
    },
    {
      "id": 25,
      "name": "pays_on"
    },
    {
      "id": 40,
      "name": "mounts_on"
    },
    {
      "id": 135,
      "name": "uses"
    }
  ]
}

```

```

    },
    {
      "id": 278,
      "name": "sells"
    }
  ],
  "isField": true,
  "typeChoices": []
},
{
  "resourcePortId": 2658,
  "resourceId": 13811,
  "portName": "Sheen",
  "direction": "out",
  "description": null,
  "isActive": true,
  "createdAt": "2026-10-17T05:33:23.552Z",
  "scope": "boundary",
  "sortOrder": 0,
  "routingRule": {},
  "required": false,
  "expectedRoleId": null,
  "derivedFromPortDefinitionId": 857,
  "types": [
    {
      "resourcePortTypeId": 786,
      "resourcePortId": 2658,
      "resourceTypeId": 1411,
      "typeName": "Sheen",
      "parameters": {}
    }
  ],
  "maxConnections": 1,
  "defaultInteractionTypeId": 135,
  "contentsScope": "inherit",
  "effectiveTypes": [
    {
      "resourceTypeId": 1411,
      "typeName": "Sheen"
    }
  ],
  "setting": null,
  "allowedInteractionTypes": [
    {
      "id": 1,
      "name": "allocates"
    },
    {
      "id": 2,
      "name": "consumes"
    },
    {
      "id": 3,
      "name": "reserves"
    },
    {
      "id": 4,
      "name": "produces"
    },
    {
      "id": 5,
      "name": "depends_on"
    },
    {
      "id": 6,
      "name": "assigned_to"
    }
  ]
}

```

```

    },
    {
      "id": 19,
      "name": "certified_in"
    },
    {
      "id": 20,
      "name": "licensed_for"
    },
    {
      "id": 21,
      "name": "derives_from"
    },
    {
      "id": 22,
      "name": "feeds"
    },
    {
      "id": 23,
      "name": "bid_for"
    },
    {
      "id": 24,
      "name": "priced_at"
    },
    {
      "id": 25,
      "name": "pays_on"
    },
    {
      "id": 40,
      "name": "mounts_on"
    },
    {
      "id": 135,
      "name": "uses"
    },
    {
      "id": 278,
      "name": "sells"
    }
  ],
  "isField": true,
  "typeChoices": []
},
{
  "resourcePortId": 2665,
  "resourceId": 13811,
  "portName": "Interior",
  "direction": "out",
  "description": null,
  "isActive": true,
  "createdAt": "2026-10-17T05:33:23.810Z",
  "scope": "boundary",
  "sortOrder": 0,
  "routingRule": {},
  "required": false,
  "expectedRoleId": null,
  "derivedFromPortDefinitionId": 860,
  "types": [
    {
      "resourcePortTypeId": 788,
      "resourcePortId": 2665,
      "resourceTypeId": 1412,
      "typeName": "Interior",
      "parameters": {}
    }
  ]
}

```

```

    }
  ],
  "maxConnections": 1,
  "defaultInteractionTypeId": 135,
  "contentsScope": "inherit",
  "effectiveTypes": [
    {
      "resourceTypeId": 1412,
      "typeName": "Interior"
    }
  ],
  "setting": null,
  "allowedInteractionTypes": [
    {
      "id": 1,
      "name": "allocates"
    },
    {
      "id": 2,
      "name": "consumes"
    },
    {
      "id": 3,
      "name": "reserves"
    },
    {
      "id": 4,
      "name": "produces"
    },
    {
      "id": 5,
      "name": "depends_on"
    },
    {
      "id": 6,
      "name": "assigned_to"
    },
    {
      "id": 19,
      "name": "certified_in"
    },
    {
      "id": 20,
      "name": "licensed_for"
    },
    {
      "id": 21,
      "name": "derives_from"
    },
    {
      "id": 22,
      "name": "feeds"
    },
    {
      "id": 23,
      "name": "bid_for"
    },
    {
      "id": 24,
      "name": "priced_at"
    },
    {
      "id": 25,
      "name": "pays_on"
    },
    {

```

```

        "id": 40,
        "name": "mounts_on"
    },
    {
        "id": 135,
        "name": "uses"
    },
    {
        "id": 278,
        "name": "sells"
    }
],
"isField": true,
"typeChoices": []
},
{
    "resourcePortId": 2671,
    "resourceId": 13811,
    "portName": "Upper Door Style",
    "direction": "out",
    "description": null,
    "isActive": true,
    "createdAt": "2026-10-17T05:33:24.072Z",
    "scope": "boundary",
    "sortOrder": 0,
    "routingRule": {},
    "required": false,
    "expectedRoleId": null,
    "derivedFromPortDefinitionId": 863,
    "types": [
        {
            "resourcePortTypeId": 790,
            "resourcePortId": 2671,
            "resourceTypeId": 3427,
            "typeName": "Upper Door Style",
            "parameters": {}
        }
    ],
    "maxConnections": 1,
    "defaultInteractionTypeId": 135,
    "contentsScope": "inherit",
    "effectiveTypes": [
        {
            "resourceTypeId": 3427,
            "typeName": "Upper Door Style"
        }
    ],
    "setting": null,
    "allowedInteractionTypes": [
        {
            "id": 1,
            "name": "allocates"
        },
        {
            "id": 2,
            "name": "consumes"
        },
        {
            "id": 3,
            "name": "reserves"
        },
        {
            "id": 4,
            "name": "produces"
        }
    ],
    {

```

```

        "id": 5,
        "name": "depends_on"
    },
    {
        "id": 6,
        "name": "assigned_to"
    },
    {
        "id": 19,
        "name": "certified_in"
    },
    {
        "id": 20,
        "name": "licensed_for"
    },
    {
        "id": 21,
        "name": "derives_from"
    },
    {
        "id": 22,
        "name": "feeds"
    },
    {
        "id": 23,
        "name": "bid_for"
    },
    {
        "id": 24,
        "name": "priced_at"
    },
    {
        "id": 25,
        "name": "pays_on"
    },
    {
        "id": 40,
        "name": "mounts_on"
    },
    {
        "id": 135,
        "name": "uses"
    },
    {
        "id": 278,
        "name": "sells"
    }
},
"isField": true,
"typeChoices": []
},
{
    "resourcePortId": 2673,
    "resourceId": 13811,
    "portName": "Lower Door Style",
    "direction": "out",
    "description": null,
    "isActive": true,
    "createdAt": "2026-10-17T05:33:24.251Z",
    "scope": "boundary",
    "sortOrder": 0,
    "routingRule": {},
    "required": false,
    "expectedRoleId": null,
    "derivedFromPortDefinitionId": 866,
    "types": [

```

```

    {
      "resourcePortTypeId": 792,
      "resourcePortId": 2673,
      "resourceTypeId": 3428,
      "typeName": "Lower Door Style",
      "parameters": {}
    }
  ],
  "maxConnections": 1,
  "defaultInteractionTypeId": 135,
  "contentsScope": "inherit",
  "effectiveTypes": [
    {
      "resourceTypeId": 3428,
      "typeName": "Lower Door Style"
    }
  ],
  "setting": null,
  "allowedInteractionTypes": [
    {
      "id": 1,
      "name": "allocates"
    },
    {
      "id": 2,
      "name": "consumes"
    },
    {
      "id": 3,
      "name": "reserves"
    },
    {
      "id": 4,
      "name": "produces"
    },
    {
      "id": 5,
      "name": "depends_on"
    },
    {
      "id": 6,
      "name": "assigned_to"
    },
    {
      "id": 19,
      "name": "certified_in"
    },
    {
      "id": 20,
      "name": "licensed_for"
    },
    {
      "id": 21,
      "name": "derives_from"
    },
    {
      "id": 22,
      "name": "feeds"
    },
    {
      "id": 23,
      "name": "bid_for"
    },
    {
      "id": 24,
      "name": "priced_at"
    }
  ]
}

```

```

    },
    {
      "id": 25,
      "name": "pays_on"
    },
    {
      "id": 40,
      "name": "mounts_on"
    },
    {
      "id": 135,
      "name": "uses"
    },
    {
      "id": 278,
      "name": "sells"
    }
  ],
  "isField": true,
  "typeChoices": []
},
{
  "resourcePortId": 2675,
  "resourceId": 13811,
  "portName": "Drawer",
  "direction": "out",
  "description": null,
  "isActive": true,
  "createdAt": "2026-10-17T05:33:24.524Z",
  "scope": "boundary",
  "sortOrder": 0,
  "routingRule": {},
  "required": false,
  "expectedRoleId": null,
  "derivedFromPortDefinitionId": 870,
  "types": [
    {
      "resourcePortTypeId": 794,
      "resourcePortId": 2675,
      "resourceTypeId": 1408,
      "typeName": "Drawer",
      "parameters": {}
    }
  ],
  "maxConnections": 1,
  "defaultInteractionTypeId": 135,
  "contentsScope": "inherit",
  "effectiveTypes": [
    {
      "resourceTypeId": 1408,
      "typeName": "Drawer"
    }
  ],
  "setting": null,
  "allowedInteractionTypes": [
    {
      "id": 1,
      "name": "allocates"
    },
    {
      "id": 2,
      "name": "consumes"
    },
    {
      "id": 3,
      "name": "reserves"
    }
  ]
}

```

```

    },
    {
      "id": 4,
      "name": "produces"
    },
    {
      "id": 5,
      "name": "depends_on"
    },
    {
      "id": 6,
      "name": "assigned_to"
    },
    {
      "id": 19,
      "name": "certified_in"
    },
    {
      "id": 20,
      "name": "licensed_for"
    },
    {
      "id": 21,
      "name": "derives_from"
    },
    {
      "id": 22,
      "name": "feeds"
    },
    {
      "id": 23,
      "name": "bid_for"
    },
    {
      "id": 24,
      "name": "priced_at"
    },
    {
      "id": 25,
      "name": "pays_on"
    },
    {
      "id": 40,
      "name": "mounts_on"
    },
    {
      "id": 135,
      "name": "uses"
    },
    {
      "id": 278,
      "name": "sells"
    }
  ],
  "isField": true,
  "typeChoices": []
},
{
  "resourcePortId": 2682,
  "resourceId": 13811,
  "portName": "Hinge",
  "direction": "out",
  "description": null,
  "isActive": true,
  "createdAt": "2026-10-17T05:33:24.812Z",
  "scope": "boundary",

```

```

"sortOrder": 0,
"routingRule": {},
"required": false,
"expectedRoleId": null,
"derivedFromPortDefinitionId": 873,
"types": [
  {
    "resourcePortTypeId": 796,
    "resourcePortId": 2682,
    "resourceTypeId": 1409,
    "typeName": "Hinge",
    "parameters": {}
  }
],
"maxConnections": 1,
"defaultInteractionTypeId": 135,
"contentsScope": "inherit",
"effectiveTypes": [
  {
    "resourceTypeId": 1409,
    "typeName": "Hinge"
  }
],
"setting": null,
"allowedInteractionTypes": [
  {
    "id": 1,
    "name": "allocates"
  },
  {
    "id": 2,
    "name": "consumes"
  },
  {
    "id": 3,
    "name": "reserves"
  },
  {
    "id": 4,
    "name": "produces"
  },
  {
    "id": 5,
    "name": "depends_on"
  },
  {
    "id": 6,
    "name": "assigned_to"
  },
  {
    "id": 19,
    "name": "certified_in"
  },
  {
    "id": 20,
    "name": "licensed_for"
  },
  {
    "id": 21,
    "name": "derives_from"
  },
  {
    "id": 22,
    "name": "feeds"
  },
  {

```

```

        "id": 23,
        "name": "bid_for"
    },
    {
        "id": 24,
        "name": "priced_at"
    },
    {
        "id": 25,
        "name": "pays_on"
    },
    {
        "id": 40,
        "name": "mounts_on"
    },
    {
        "id": 135,
        "name": "uses"
    },
    {
        "id": 278,
        "name": "sells"
    }
],
"isField": true,
"typeChoices": []
},
{
    "resourcePortId": 2689,
    "resourceId": 13811,
    "portName": "Slide",
    "direction": "out",
    "description": null,
    "isActive": true,
    "createdAt": "2026-10-17T05:33:25.074Z",
    "scope": "boundary",
    "sortOrder": 0,
    "routingRule": {},
    "required": false,
    "expectedRoleId": null,
    "derivedFromPortDefinitionId": 876,
    "types": [
        {
            "resourcePortTypeId": 798,
            "resourcePortId": 2689,
            "resourceTypeId": 1410,
            "typeName": "Slide",
            "parameters": {}
        }
    ],
    "maxConnections": 1,
    "defaultInteractionTypeId": 135,
    "contentsScope": "inherit",
    "effectiveTypes": [
        {
            "resourceTypeId": 1410,
            "typeName": "Slide"
        }
    ],
    "setting": null,
    "allowedInteractionTypes": [
        {
            "id": 1,
            "name": "allocates"
        }
    ],
    {

```

```

        "id": 2,
        "name": "consumes"
    },
    {
        "id": 3,
        "name": "reserves"
    },
    {
        "id": 4,
        "name": "produces"
    },
    {
        "id": 5,
        "name": "depends_on"
    },
    {
        "id": 6,
        "name": "assigned_to"
    },
    {
        "id": 19,
        "name": "certified_in"
    },
    {
        "id": 20,
        "name": "licensed_for"
    },
    {
        "id": 21,
        "name": "derives_from"
    },
    {
        "id": 22,
        "name": "feeds"
    },
    {
        "id": 23,
        "name": "bid_for"
    },
    {
        "id": 24,
        "name": "priced_at"
    },
    {
        "id": 25,
        "name": "pays_on"
    },
    {
        "id": 40,
        "name": "mounts_on"
    },
    {
        "id": 135,
        "name": "uses"
    },
    {
        "id": 278,
        "name": "sells"
    }
    ],
    "isField": true,
    "typeChoices": []
}
]

```

The capture above shows two fields holding their own values. For a field that inherits its value, the row's `setting` looks like this (illustrative; names are examples):

```
"setting": {
  "value": { "resourceId": <laminateId>, "name": "Laminate" },
  "scope": "inherited",
  "locked": true,
  "source": { "resourceId": <setId>, "name": "Kitchen design set" },
  "overrides": null,
  "conflicts": [],
  "note": "Laminate · locked by Kitchen design set",
  "clear": null,
  "deviates": false
}
```

The other `note` forms are "`<value> · from <source>`" (inherited, not locked), "`Overrides <value> from <source>`" (an own value replacing an inherited one), "`Set here · same as <source>`" (an own value equal to the inherited one it covers — kept so it stays put if the enclosing item changes), "`Conflicts with <value>, locked by <source>`" (an own value that differs from a lock above it) and "`<value> from <source> · <value> from <source> – enclosing items disagree`" (enclosing items pass down different values). The `clear` label reads "`Clear – use <value> from <source>`", "`Clear – use <value>, locked by <source>`" or "`Clear`" (nothing would apply).

Errors: 400 `invalid_resource_id`, 403 `subject_outside_grant`, 403 `type_not_readable`, 404 `resource_not_found`.

GET /v1/resources/:id/ports/:portId/legal-targets

For one port, which resources it could connect to now, and why not where it can't. **Needs read scope** on the resource (and on the context, for `scope=context`).

Parameter	Type	Notes
<code>scope</code>	string	<code>type</code> — every resource of the accepted types you can read; <code>context</code> (default) — the resources in <code>contextId</code> you can read.
<code>contextId</code>	string	Required for <code>scope=context</code> .
<code>excludeInteractionId</code>	string	Judge as if this existing connection on the port were gone (to change a choice).
<code>q</code>	string	Only candidates whose name or identifier contains this (case-insensitive). Either <code>scope</code> .
<code>limit</code>	number	Optional, 1–100. Return at most this many candidates, newest first (by creation, then id). Out of range is 400 <code>invalid_limit</code> .

Response: { `sourcePort`, `candidates`: [{ `resourceId`, `name`, `resourceTypeId`, `verdict`, `reasons`, `viableTargetPorts` }] }.

Without `limit` the list is complete, in no promised order for `scope=type` and in the context's own order for `scope=context`, and the response carries nothing else. **With `limit`** it also carries `truncated` (`true` when candidates were left out) and `total` (how many candidates there are in all, after `q`). Every candidate returned carries its own verdict either way. `q` with a small `limit` — a line's `product` field narrowed to one identifier:

From `legal-targets-q.json` — status 200:

```
{
  "sourcePort": {
    "portId": 7736,
    "portName": "product",
    "direction": "out",
    "maxConnections": 1,
    "held": 0,
    "capRemaining": 1
  },
  "candidates": [
    {
      "resourceId": 9009,
      "name": "AP006",
      "resourceTypeId": 182,
      "verdict": "legal",
      "reasons": [],
      "viableTargetPorts": [
        {
          "portId": 4606,
          "portName": "Sold as",
          "direction": "in"
        }
      ]
    }
  ],
  "truncated": false,
  "total": 1
}
```

Without `q`, over the whole price book:

From `legal-targets-limit.json` — status 200:

```

{
  "sourcePort": {
    "portId": 4372,
    "portName": "product",
    "direction": "out",
    "maxConnections": 1,
    "held": 0,
    "capRemaining": 1
  },
  "candidates": [
    {
      "resourceId": 4693,
      "name": "WOCM3",
      "resourceTypeId": 37,
      "verdict": "legal",
      "reasons": [],
      "viableTargetPorts": [
        {
          "portId": 4139,
          "portName": "Sold as",
          "direction": "in"
        }
      ]
    },
    {
      "resourceId": 4692,
      "name": "OCM3",
      "resourceTypeId": 37,
      "verdict": "legal",
      "reasons": [],
      "viableTargetPorts": [
        {
          "portId": 4138,
          "portName": "Sold as",
          "direction": "in"
        }
      ]
    },
    {
      "resourceId": 4691,
      "name": "SH0E4-8",
      "resourceTypeId": 37,
      "verdict": "legal",
      "reasons": [],
      "viableTargetPorts": [
        {
          "portId": 4137,
          "portName": "Sold as",
          "direction": "in"
        }
      ]
    },
    {
      "resourceId": 4690,
      "name": "SH0E3-8",
      "resourceTypeId": 37,
      "verdict": "legal",
      "reasons": [],
      "viableTargetPorts": [
        {
          "portId": 4136,
          "portName": "Sold as",
          "direction": "in"
        }
      ]
    }
  ],
}

```

```

{
  "resourceId": 4689,
  "name": "WSCM8",
  "resourceTypeId": 37,
  "verdict": "legal",
  "reasons": [],
  "viableTargetPorts": [
    {
      "portId": 4135,
      "portName": "Sold as",
      "direction": "in"
    }
  ]
},
"truncated": true,
"total": 3925
}

```

`verdict` is `legal` or `illegal`; each reason is `{ error, message }` with the same codes a create would return (`accepted_type`, `compat_deny`, `contents_locked`, `port_max_connections`, `port_inactive`), plus the advisory `no_viable_target_port`. A design set's Finish, narrowed to a value its Species rules out:

From `legal-targets-type.json` — status 200:

```

{
  "sourcePort": {
    "portId": 4212,
    "portName": "Finish",
    "direction": "out",
    "maxConnections": 1,
    "held": 0,
    "capRemaining": 1
  },
  "candidates": [
    {
      "resourceId": 371,
      "name": "Worldly Gray (SW7043)",
      "resourceTypeId": 119,
      "verdict": "illegal",
      "reasons": [
        {
          "error": "compat_deny",
          "message": "Incompatible with Outsource Specie (Species).",
          "denyEdgeId": 8,
          "layer": "default",
          "firingNeighborId": 298,
          "firingNeighborName": "Outsource Specie",
          "firingNeighborTypeName": "Species"
        }
      ],
      "viableTargetPorts": [
        {
          "portId": 102,
          "portName": "Used by",
          "direction": "in"
        }
      ]
    }
  ],
  "truncated": false,
  "total": 1
}

```

A port is listed in `viableTargetPorts` only when the connection faces the right way: the edge's source port sends (`out` or `bidirectional`) and its target port receives (`in` or `bidirectional`). An `out` port is the source of the connection it starts, an `in` port the target. A create that faces the wrong way is refused with `422 port_direction`.

A connection whose source sits in a read-only status is never listed: the candidate is `illegal` with the reason `status_read_only` — the same code a create returns (`409`). When the anchor itself is the read-only source, every candidate carries it.

A `compat_deny` reason may also say what fired it:

Field	Type	Notes
<code>denyEdgeId</code>	number null	The compatibility rule that refused the pair.
<code>layer</code>	string	Where that rule was found — on the values themselves or a default for their kinds.
<code>firingNeighb orId</code>	number null	When the refusal comes from something already attached to the anchor, that resource. <code>null</code> when the anchor and the candidate refuse each other directly, or when you cannot read the neighbour.
<code>firingNeighb orName</code>	string null	Its name, under the same rule (<code>null</code> with the id).
<code>firingNeighb orTypeName</code>	string null	Its type's name — <code>null</code> when you cannot read the type.

A refusal that comes from a setting further up or further down the containment may instead carry `viaAncestorId`, `offendingResourceId`, `offenderCount`, `kindTypeId` or `lockEdgeId`. Any of these ids is left out when you cannot read that resource; the message then says `an enclosing item` or `an item inside`.

Errors: `400 invalid_resource_id`, `400 context_id_required`, `400 invalid_context_id`, `400 exclude_mismatch`, `400 invalid_limit`, `403` as above, `404 port_not_found`.

Note — not available to keys. `GET /v1/resources/:id/compat-candidates` (the resources a person may mark compatible or incompatible with this one, each with the verdict and reasons a marking would get and a sentence for the pair's current state) is served to signed-in people only, like compatibility marking itself. A key gets `403 compat_authoring_session_only` there.

POST /v1/resource-interactions

Make an attachment. **Needs operate scope on the choosing side, read scope on the chosen side.**

Field	Type	Notes
sourceResourceId	number	Required.
targetResourceId	number	Required.
interactionTypeId	number	Required. One of the source port's <code>allowedInteractionTypes</code> (see <code>GET /v1/resources/:id/ports</code>).
sourcePortId	number	Required. A port on the source.
targetPortId	number	Required. A port on the target.
replaceInteractionId	number	Optional. Swap an existing connection on the same source port for this one, atomically.

Returns 201 { `interactionId`, `sourceResourceId`, `targetResourceId`, `interactionTypeId`, `contextId` } — here a bid's `customer` field:

From `bid-field-customer.json` — status 201:

```
{
  "interactionId": 2720,
  "sourceResourceId": 24911,
  "targetResourceId": 24910,
  "interactionTypeId": 23,
  "contextId": null
}
```

From `interaction-refused-compat.json` — status 422:

```
{
  "error": "compat_deny",
  "message": "Incompatible with Outsource Specie (Species).",
  "denyEdgeId": 136,
  "layer": "default",
  "a": 12940,
  "b": 7272,
  "via": 24918
}
```

From `interaction-not-operable.json` — status 403:

```
{
  "error": "type_not_operable"
}
```

Errors: 400 `source_resource_id_required`, 400 `target_resource_id_required`, 400 `interaction_type_id_required`, 400 `self_interaction`, 400 `ports_required`, 400 `port_not_found`, 400 `port_not_on_resource`, 400 `replace_mismatch`, 403 `subject_outside_grant`, 403 `type_not_operable`, 403 `type_not_readable`, 403 `compat_authoring_session_only`, 409

`status_read_only` (the choosing side sits in a read-only status — a frozen line cannot change its product), `422 port_max_connections`, `422 port_inactive`, `422 accepted_type`, `422 compat_deny`, `422 contents_locked`, `422 compat_type_not_connectable`, `422 port_direction`.

A connection type with no ports — `derives_from`, the link between a resource and the one it was made from — cannot be drawn here:

From `interaction-400-ports-required.json` — status 400:

```
{
  "error": "ports_required",
  "message": "sourcePortId and targetPortId are required"
}
```

Two requests at once. When two requests try to fill a port that takes one connection at the same moment, one succeeds and the other gets `422 port_max_connections`. A `replaceInteractionId` that another request has already replaced gets `400 replace_mismatch`.

DELETE `/v1/resource-interactions/:id`

Remove an attachment. **Same scopes as creating it.** Returns `204` with no body (`interaction-delete.json`). Compatibility markings cannot be removed by a key. Deleting a connection that is already gone — including one another request deleted a moment earlier — returns `404 interaction_not_found`, and only the deletion that happened is recorded.

Errors: `400 invalid_interaction_id`, `403` as above, `403 compat_authoring_session_only`, `404 interaction_not_found`.

Creating and addressing resources

POST `/v1/resources/find-or-create`

Find a resource by an identifier from **your** system, or create it if it is not there. This is the idempotent way to bring your records into ReSync: call it again with the same key and you get the same resource back, not a second one.

Needs the type to be in `find_or_create_types`.

This body is snake_case, unlike everywhere else in the API. It predates the rest of this surface and has not been changed, because renaming the fields would break the integrations already using it.

Field	Type	Notes
<code>type_key</code>	string	Required. The type code to create.
<code>external_system</code>	string	Required. The name of an external system ReSync already knows — your administrator tells you which is yours. Identifiers are unique per system across every organization, so prefix yours.
<code>external_key</code>	string	Required. Your identifier for this thing.
<code>properties</code>	object	Optional. Applied at creation only — ignored if the resource already exists.
<code>name</code>	string	Optional. The new resource's name. A type that computes its names from properties ignores it; otherwise it wins over <code>properties.name</code> , and with neither the type's code is used. A blank <code>name</code> is <code>400 invalid_resource_update</code> . Ignored if the resource already exists.
<code>actorResourceId / actorAlias</code>		Optional, as for <code>PATCH</code> (these two keep their camelCase names). Without them the birth is attributed to your key's own service identity.

One step, all or nothing. The resource, your identifier for it and its `resource.created` event are written together. `resource.created` carries the `properties` it was created with and `alias: { system, code }` — your identifier — so its history starts complete; no separate `property.changed` follows. If two calls race on the same identifier, one creates and the other gets that same resource back (`200`). A new resource is then placed by any enrolment rules its type declares, as if a person had created it.

Created:

From `foc-create.json` — status 201:

```
{
  "resource_id": 24908,
  "created": true
}
```

Called again with the same identifier:

From `foc-find.json` — status 200:

```
{
  "resource_id": 24908,
  "created": false
}
```

What is refused — all as `find_or_create_refused`, with the reason in `message`:

- `403` — the type is not in your `find_or_create_types`;

- `400` — the external system is not registered, or is `resync` (ReSync's own identifiers are not an external system), or the type is a built-in system type;
- `409` — the identifier is already bound to a resource of **another type**: the message says so and names no resource. Use a different identifier, or ask for the type you actually hold;
- `409` — the identifier is bound to a resource your key cannot read. That resource is not named — it does not exist for you, and a new one cannot take its identifier.

From `foc-refused-resync.json` — status 400:

```
{
  "error": "find_or_create_refused",
  "message": "find-or-create is for external systems; 'resync' is the canonical system"
}
```

From `foc-type-mismatch.json` — status 409:

```
{
  "error": "find_or_create_refused",
  "message": "kpc-line-1 is already bound under pdmac-sql to a resource of another type"
}
```

Errors: `401 invalid_api_key`, `400 / 403 / 409 find_or_create_refused`, `400 invalid_resource_update` (blank name), `403 owner_grant_required` (a key with no owning organization), `403 actor_not_readable`, `409 unique_key_taken` (the new resource's `properties` would take a value a declared unique key already holds).

Creating a resource and what it holds

The same door can create a resource **together with what it holds** — its fields and the resources inside it — in one request. Add either or both of these to the body; each child has the same shape again, so a whole bid (its design set, the set's lines, a line's modification, the installation line) is one body:

Field	Type	Notes
<code>fields</code>	object	Optional. { " <code>portName</code> ": <resource id> } — the resource each field points at, by the field's <code>portName</code> (as GET <code>/v1/resources/:id/ports</code> names it). The connection uses the field's own interaction type; to name a different one, give { " <code>target</code> ": <resource id>, " <code>interaction_type_id</code> ": <id> } instead of the bare id.
<code>children</code>	array	Optional. Resources to put inside this one, each with <code>type_key</code> , <code>external_system</code> , <code>external_key</code> and, as above, <code>name</code> , <code>properties</code> , <code>fields</code> , <code>children</code> . A child may also carry <code>role_id</code> (as <code>componentRoleId</code> on POST <code>/v1/resources/:id/members</code>).

Every resource in the body follows the find-or-create rules above: its type must be in `find_or_create_types`, and it is found by your identifier or created. The whole request is **one step, all or nothing**: if any part is refused, nothing in it is created, put inside anything, or connected.

The answer mirrors the body: each resource's `resource_id` and `created`, and its `children` in the order you sent them. The status is `201` if anything was created, `200` if everything already existed. A body with neither `fields` nor `children` answers exactly as above, { `resource_id`, `created` }.

Created — a bid, its customer, level and terms, a design set with its ten fields, five lines with their products (a modification under the first), and an installation line:

From `foc-tree-create.json` — status 201:

```
{
  "resource_id": 24925,
  "created": true,
  "children": [
    {
      "resource_id": 24926,
      "created": true,
      "children": [
        {
          "resource_id": 24927,
          "created": true,
          "children": [
            {
              "resource_id": 24928,
              "created": true,
              "children": []
            }
          ]
        },
        {
          "resource_id": 24929,
          "created": true,
          "children": []
        },
        {
          "resource_id": 24930,
          "created": true,
          "children": []
        },
        {
          "resource_id": 24931,
          "created": true,
          "children": []
        },
        {
          "resource_id": 24932,
          "created": true,
          "children": []
        }
      ]
    },
    {
      "resource_id": 24933,
      "created": true,
      "children": []
    }
  ]
}
```

Sending it again finds every resource by its identifier and changes nothing. A resource that already exists keeps its name and properties; a child already inside its parent stays as it is; a field that already points at the resource you name is left alone. So a request that timed out can be resent as it was:

From `foc-tree-replay.json` — status 200:

```

{
  "resource_id": 24925,
  "created": false,
  "children": [
    {
      "resource_id": 24926,
      "created": false,
      "children": [
        {
          "resource_id": 24927,
          "created": false,
          "children": [
            {
              "resource_id": 24928,
              "created": false,
              "children": []
            }
          ]
        },
        {
          "resource_id": 24929,
          "created": false,
          "children": []
        },
        {
          "resource_id": 24930,
          "created": false,
          "children": []
        },
        {
          "resource_id": 24931,
          "created": false,
          "children": []
        },
        {
          "resource_id": 24932,
          "created": false,
          "children": []
        }
      ]
    },
    {
      "resource_id": 24933,
      "created": false,
      "children": []
    }
  ]
}

```

A field that already points somewhere **else** is not changed by a resend: it is refused like any second choice on a field (422 port_max_connections). To change a field, use `replaceInteractionId` on `POST /v1/resource-interactions`.

When one part is refused, the answer is that part's own error — the same status, `error` and `message` as the single-resource door would give — with one extra field, `item: path` is where that resource sits in your body (`children[0].children[2]` is the third child of the first child; the top resource has no `path`), and `external_key` is your identifier for it. Here line three's product field points at a design choice, not a product:

From `foc-tree-refused-item.json` — status 422:

```
{
  "error": "accepted_type",
  "message": "This port does not accept resources of type Species.",
  "portId": 7928,
  "endpoint": "source",
  "offendingResourceId": 7272,
  "offendingTypeId": 1404,
  "item": {
    "path": "children[0].children[2]",
    "external_key": "docs-tree-bad-line-3"
  }
}
```

A refusal's `message` does not name a resource created earlier in the same request — the request has not finished, so it is not yet yours to read by name; `item` tells you which part failed.

Limits. At most 250 resources in one body, counting the top one, and at most 8 levels deep. Past either, the request is `400 find_or_create_refused` before anything is read. The whole body is one request against your rate limit.

Errors, in addition to the ones above: any error of `POST /v1/resources/:id/members` (for a child) or of `POST /v1/resource-interactions` (for a field) — `422 accepted_type`, `422 port_max_connections`, `422 containment_cap`, `422 compat_deny`, `409 status_read_only` and the rest — each with `item`; `400 port_not_found` (no field of that name on that type); `400 interaction_type_id_required` (a field that declares no interaction type and names none).

POST /v1/aliases/bind

Attach one of your identifiers to a resource that already exists. Also `snake_case`.

Field	Type	Notes
<code>external_system</code>	string	Required.
<code>external_key</code>	string	Required.
<code>resource_id</code>	number	Required.

The target must be a resource your key may change (it is in your organization and its type is in `operate_types`). Idempotent to the same resource; `409` if the identifier is already bound to a different one — that resource is not named. Needs `bind_systems` to include the system (see [auth-and-scopes.md](#)).

From `aliases-bind.json` — *status 201*:

```
{
  "resource_id": 24911,
  "created": true
}
```

Errors: `401 invalid_api_key`, `403 subject_outside_grant`, `400 / 404 / 409` as `alias_bind_refused` with the reason in `message`.

GET /v1/aliases/resolve

Look up the resource behind one of your identifiers. Read-only; never creates anything.

Parameter	Type	Notes
system	string	Required.
code	string	Required.

Response: { resourceId, resourceType, name } — resourceType is the type's display **name**, not its code (key on the code from GET /v1/resources/:id instead). **Needs read scope** on the resource it resolves to; an identifier bound to a resource your key cannot read answers exactly like one bound to nothing.

From aliases-resolve.json — status 200:

```
{
  "resourceId": 24911,
  "resourceType": "Bid",
  "name": "Harbor Homes – kitchen"
}
```

Errors: 401 invalid_api_key, 400 system_and_code_required, 400 unknown_external_system, 404 alias_not_found.

POST /v1/identity-claims/redeem

For a **bridge** — for example a messaging bot — that links an external account to a person in ReSync. The person creates a claim in the ReSync app and receives an 8-character code; they send it to your bridge, and your bridge redeems it, naming the external account. **Key-only:** a signed-in person is refused 403.

Field	Type	Notes
code	string	Required. The 8-character claim code.
actorAlias	object	Required. { system, code } — the external account being linked. It is found, or created under your organization the first time it is seen.

Needs bind_systems to include actorAlias.system — linking binds an identifier under that system, exactly as POST /v1/aliases/bind does. This is checked before the code is looked at, so a key without it learns nothing about pending claims.

Response: 200 { status: "linked", identityLinkId, stubResourceId, canonicalResourceId }, or 200 { status: "already_linked", ... } when that account is already linked to the same person.

From identity-claims-no-bind-systems.json — status 403:

```
{
  "error": "alias_bind_refused",
  "message": "key not scoped to bind under 'telegram'"
}
```

Errors: 401 invalid_api_key, 403 alias_bind_refused. Its other refusals put a sentence in error: 400 (no code, or no actorAlias), 404 (the code is unknown or expired), 409 (the account is already a verified account, is linked to someone else, or the person has a pending invite). Match on the HTTP status for those.

Reference data

These describe how the world is modelled. The lists are not type-scoped — any valid key may read them — and they change rarely, so cache them for the life of your process. **A by-id read follows the list:** `GET /v1/resource-types/:id`, `GET /v1/lifecycles/:id`, `GET /v1/lifecycles/:id/graph` and `GET /v1/state-transitions?lifecycleId=` answer `404` for anything your key's own list would not show, exactly as if it did not exist.

Endpoint	Returns
GET /v1/resource-types	Types available to your organization, each with its <code>code</code> and <code>resourceTypeId</code> . Optional <code>lifecycleId</code> filter. A full capture is <code>resource-types-list.json</code> in the fixture set.
GET /v1/resource-types/:id	One type in full, including its declared property schema and <code>containmentCaps</code> — the limits it declares as a member ([{ <code>containmentCapId</code> , <code>parentTypeId</code> , <code>maxParents</code> }] : a resource of this type, or of a type under it, can be inside at most <code>maxParents</code> containers whose type is <code>parentTypeId</code> or a type under it). Captured in the quickstart (<code>resource-type-by-id.json</code>).
GET /v1/statuses	Every status, with the lifecycle it belongs to.
GET /v1/lifecycles	Every lifecycle.
GET /v1/lifecycles/:id	One lifecycle with its statuses and transitions.
GET /v1/lifecycles/:id/graph	One lifecycle as a graph in a single read: its statuses in display order (from the entry status outward, unreachable ones last), its transitions with their requirements and gate counts, the types that run it, and the consequence rules on those types — each rule listing the transitions whose required evidence is the event that triggers it. Every label is ready to display. <code>lifecycle.can.author</code> is always <code>false</code> for a key. Each status also carries <code>legalTargets</code> (the statuses a new transition from it may reach) and <code>illegalTargets</code> (every other status of the lifecycle, mapped to the sentence that says why not); both are empty unless the reader can author the lifecycle, so they are always empty for a key. Each transition carries <code>capture</code> (what it photographs when it fires — see <code>resource.captured</code> in events.md ; <code>null</code> when nothing) only for a reader who can author the lifecycle, so never for a key.
GET /v1/state-transitions	Legal moves. Needs <code>fromStatusId</code> plus <code>resourceId</code> or <code>lifecycleId</code> . With <code>resourceId</code> , your key must be able to read that resource — refusals as for <code>GET /v1/resources/:id</code> .
GET /v1/attestation-event-types	The <code>eventType</code> values you may attest, with their declared fields. A full capture is <code>attestation-event-types-read.json</code> in the fixture set.

Endpoint	Returns
GET /v1/property-keys	Property keys in use across types, with labels and value types.
GET /v1/units	Units of measure.
GET /v1/interaction-types	The kinds of relationship that can exist between resources. Each carries <code>connectable : true</code> when it may be used for a connection, <code>false</code> for compatibility types, which are set from a resource's compatibility sheet rather than as connections.

Deprecated, informational only — will be removed in a future revision, with notice here first; do not branch on them: `isVerified` on resource reads (always false today) and `affectsInventory` / `affectsAvailability` / `affectsScheduling` on interaction-type reads (declared, never acted on).

The legal moves for one of your resources:

From `state-transitions-read.json` — status 200:

```
{
  "transitions": [
    {
      "transitionId": 824,
      "fromStatusId": 1318,
      "toStatusId": 1319,
      "actionLabel": "Freeze Line",
      "autoAdvance": false,
      "toStatusName": "Frozen",
      "toStatusPreventsUsage": false,
      "toStatusIsAllocating": false,
      "toStatusSubjectReadOnly": true
    }
  ]
}
```

A key with no read or operate scope at all can still read these, but will be refused everywhere else — so if reference data works and nothing else does, your scopes are the thing to check.

7 Errors

The envelope

Errors carry a machine-readable code, an optional human sentence, and sometimes structured detail:

```
{ "error": "type_not_operable" }
```

```
{
  "error": "invalid_include",
  "message": "unknown include member(s): foo (accepted: properties)"
}
```

Branch on `error`. It is a stable string and will not change within v1. `message` is for your logs and your support tickets; it may be reworded at any time. Extra fields (`keys` , `requirements` , `retryAfterSeconds` , `item`) carry detail specific to that code. `item` appears only when a `find-or-create` that carries `fields` or `children` is refused: it names the part of your body that failed — `{ "path": "children[0].children[2]", "external_key": "..." }` (no `path` for the top resource) — beside that part's own `error` and `message`.

Two shapes you will also meet

Authentication failures on most endpoints return a different shape:

```
{ "statusCode": 401, "error": "Unauthorized", "message": "Invalid token" }
```

Here `error` is a phrase, not a code. This happens because those endpoints accept either a key or a signed-in person, and the rejection is raised before the key layer sees it. A missing key and a malformed or unknown key are indistinguishable this way.

A revoked key is the exception. If the value you send is a key that has been revoked, these endpoints return the normal envelope with `invalid_api_key`, so you can tell a revoked credential from a typo:

From `dual-auth-revoked-key.json` — status 401:

```
{
  "error": "invalid_api_key",
  "message": "This api key has been revoked"
}
```

Treat any 401 as "check your key" and do not try to parse it further. The key-only endpoints (`/v1/api-keys/self` , `/v1/aliases/*` , `/v1/resources/find-or-create` , `/v1/presence-sessions*` , `/v1/alias-enrollments/redeem`) return the normal envelope with `error: "invalid_api_key"`.

Some doors return sentences instead of codes for their own validation. Their scope refusals (`403 subject_outside_grant` , `type_not_readable` , `type_not_operable` ...) are always proper codes, but these checks put an English sentence in `error`:

- `POST /v1/resources/:id/attestations` — an unknown `eventType`, a non-flat `payload`, a missing declared field;
- `PATCH /v1/resources/:id/status` — an illegal move, for example "Resource is already in this status" or "Transition blocked by unmet requirements" (read requirements in the 422 body for the detail);
- `GET /v1/state-transitions` — a missing `fromStatusId`, or neither `resourceId` nor `lifecycleId`;
- `GET /v1/resources/:id/requirements` — a missing or malformed `transitionId`;
- `GET /v1/resources/:id/members` — a malformed id; `GET /v1/resource-types/:id` — a malformed id, or a type your key's own list does not show (404);
- `POST /v1/identity-claims/redeem` — a missing code or `actorAlias`, an unknown or expired code, an account that cannot be linked;
- **the actor fields** on every write that accepts them (`PATCH`, `members`, `exit`, `move`, `attestations`, `status`, `find-or-create`) — for example "actorResourceId and actorAlias are mutually exclusive" or "actorResourceId must reference an existing person resource".

Match on the HTTP status for those, not on the string.

Every code

Authentication — fix your credential

Code	Status	Meaning	What to do
<code>invalid_api_key</code>	401	No key, unknown key, or a revoked key. On endpoints that also accept a signed-in person, only a revoked key returns this code; a missing or unknown one returns the phrase shape above.	Check the header. If it was working, ask whether the key was revoked. Do not retry.
<code>missing_bearer_token</code>	401	No <code>Authorization</code> header at all.	Fix the request.

Scope — ask your administrator

These mean your key is working correctly and is not permitted to do this. **Retrying will not help.**

Code	Status	Meaning	What to do
owne r_gra nt_re quire d	403	Your key has no owning organization. Any door that scopes to it can return this — the lists, the per-resource reads and writes, and <code>find-or-create</code> , which would otherwise have no organization to create the resource in.	Ask for a correctly provisioned key.
read _type s_req uire d	403	Your key has no readable types at all.	Ask for read scope.
oper ate_t ypes_ requi red	403	Your key has no writable types at all.	Ask for operate scope.
type _not_ reada ble	403	This type is in neither of your sets.	Ask for it to be added to <code>read_types</code> .
type _not_ opera ble	403	You may read this type but not change it.	Ask for it to be moved to <code>operate_types</code> .
subj ect_o utsid e_gra nt	403	The resource belongs to another organization.	Stop. This is the tenancy boundary; it is not a scope you can be granted.
verb _not_ grant ed	403	You may reach this resource, but not with this kind of action. A refusal that comes from your key's own scopes is always reported as <code>type_not_readable</code> , <code>type_not_operable</code> or <code>outside_acts_at_scope</code> ; this code is returned only when the limit is on the side of the person who provisioned your key — they reach the resource through access granted to them directly, or through a connection shared by another organization, and that access does not include this action (or no longer does). A key never holds more than its provisioner.	Ask for the action to be granted to the person who provisioned the key.
pers onal_ model ing_o ff	403	The resource is owned by a person rather than an organization, and this instance has personal modeling turned off: anything a person owns can be read, but not changed. A person's own record is not affected.	Stop. Work on resources your organization owns.
key_ unbou nd	403	Your key has not been bound to an acting identity. Only a key provisioned outside the normal path can be in this state.	Ask your administrator to re-provision the key.

Code	Status	Meaning	What to do
actor_not_readable	403	You named an acting person (<code>actorResourceId</code>) your key cannot read. A key may only attribute its action to someone within its reach.	Send an <code>actorAlias</code> instead, or leave the actor out — the action is then attributed to your key.
live_key_not_supported	403	Live updates (<code>GET /v1/live</code>) are held by signed-in people and device credentials only; an integration key cannot open a stream.	Poll the resource's events instead (<code>GET /v1/resources/:id/events</code>) with a stored cursor.
owner_forced_to_grant	403	You passed an <code>ownerResourceId</code> that is not yours.	Drop the parameter — the filter is applied for you.
outside_acts_within_scope	403	Your key is a device credential (it acts at one resource, such as a workstation) and the resource is not at that place. An integration key never receives this.	Act on a resource at the station, or use an integration key.

Request — fix the request

Code	Status	Meaning
<code>invalid_resource_id</code>	400	The id in the path is not a number.
<code>invalid_type_id</code>	400	<code>typeId</code> is not a number.
<code>invalid_status_id</code>	400	<code>statusId</code> is not a number.
<code>invalid_owner_resource_id</code>	400	<code>ownerResourceId</code> is not a number.
<code>invalid_cursor</code>	400	<code>cursor</code> is not a cursor this API issued.
<code>invalid_tag_id</code>	400	A <code>tagId</code> on <code>GET /v1/resources</code> is not a number. Every repeated value is checked; one bad value refuses the request.
<code>sort_not_supported</code>	400	<code>GET /v1/resources</code> with <code>sort</code> from a key. A key list is always in <code>resourceId</code> order, because its <code>cursor</code> is the last <code>resourceId</code> and could not resume a name or date order exactly. <code>message</code> says so. Drop <code>sort</code> and order the rows yourself.
<code>invalid_include</code>	400	An <code>include</code> member that does not exist. <code>message</code> lists what is accepted.
<code>type_required</code>	400	A list without <code>typeKey</code> or <code>typeId</code> .
<code>child_resource_id_required</code>	400	Enrolment without <code>childResourceId</code> .
<code>container_id_required</code>	400	Exit without <code>containerId</code> , or a move without both <code>fromContainerId</code> and <code>toContainerId</code> (each a positive integer).
<code>invalid_component_role_id</code>	400	<code>componentRoleId</code> is not a positive integer.
<code>self_membership</code>	400	A resource cannot contain itself.
<code>role_type_mismatch</code>	400	That role belongs to a different type than the container.
<code>system_and_code_required</code>	400	Alias lookup without <code>system</code> and <code>code</code> .
<code>unknown_external_system</code>	400	That external system is not registered.
<code>source_resource_id_required</code>	400	A connection without <code>sourceResourceId</code> .

Code	Status	Meaning
target_resource_id_required	400	A connection without <code>targetResourceId</code> .
interaction_type_id_required	400	A connection without <code>interactionTypeId</code> .
self_interaction	400	A resource cannot connect to itself.
ports_required	400	A connection needs both <code>sourcePortId</code> and <code>targetPortId</code> .
port_not_on_resource	400	A port you named belongs to a different resource.
invalid_interaction_id	400	The connection id in the path is not a number.
context_id_required	400	<code>legal-targets</code> with <code>scope=context</code> but no <code>contextId</code> .
invalid_context_id	400	<code>contextId</code> is not a number.
invalid_limit	400	<code>legal-targets</code> <code>limit</code> is not a whole number from 1 to 100. <code>limit</code> is optional: leave it out for the complete list. (The resource lists clamp their <code>limit</code> instead.)
invalid_enroll_into	400	GET <code>/v1/resources/search</code> <code>enrollInto</code> is not an id.
invalid_move_of	400	GET <code>/v1/resources/search</code> <code>moveOf</code> is not an id.
invalid_move_from	400	GET <code>/v1/resources/search</code> <code>moveFrom</code> is not an id.
invalid_resource_update	400	A <code>name</code> that is empty after trimming — on PATCH <code>/v1/resources/:id</code> or <code>find-or-create</code> . <code>message</code> says so.
replace_mismatch	400	POST <code>/v1/resource-interactions</code> with <code>replaceInteractionId</code> : that connection does not exist, or does not start at the same resource and port as the new one. <code>message</code> says so.
exclude_mismatch	400	<code>excludeInteractionId</code> is not a connection on that resource and port.
invalid_workspace	400	<code>workspace</code> is not an id.
workspace_not_in_scope	400	<code>workspace</code> names an owner outside your key's reach.

Not found

Code	Status	Meaning
port_not_found	404 / 400	The port is not on that resource (<code>legal-targets</code> : 404), or a port id you sent does not exist (create: 400).
interaction_not_found	404	No connection has that id.
resource_not_found	404	No resource with that id.
no_acting_resource	404	<code>GET /v1/api-keys/self/view</code> : your key does not act at a resource, so there is nothing to show.
type_not_found	404	No type with that code or id.
role_not_found	404	No role with that id.
alias_not_found	404	No resource bound to that identifier — or it is bound to a resource your key cannot read. An identifier you cannot read does not exist for you; the two cases are deliberately indistinguishable.

Note the difference between `404 resource_not_found` and `403 subject_outside_grant` : the first means nothing exists with that id, the second means something does and it is not yours.

Conflict — the current state refuses

Code	Status	Meaning	What to do
<code>already_a_member</code>	409	Already inside that container — on an enrolment, or on a move whose destination already holds it. A move whose two containers are the same also returns this.	Treat as success if you were making it so.
<code>not_active_member</code>	409	Not inside that container — on an exit, or on a move from a container it is not in.	Same.
<code>confirmed_key_locked</code>	409	A property in your patch was confirmed by a person. <code>keys</code> lists which.	Drop those keys and resend. A person must re-confirm before the value can change.
<code>unique_key_taken</code>	409	The resource's type (or a parent type) declares a unique key — the <code>identifier</code> , or one property — and another live resource of the same owner already holds that value. <code>message</code> says so in a sentence naming the owner; <code>holderResourceId</code> , <code>key</code> and <code>value</code> say which resource holds it. Raised by any create, find-or-create, property or identifier change, or unarchive.	Use a different value, or work with the holder. An archived resource releases its key.
<code>status_read_only</code>	409	The resource sits in a status its lifecycle declares read-only (for example a sent bid). Its own state cannot change: properties, name, identifier, the members it contains, the connections it is the source of, its ports, and un-confirming a property. <code>message</code> says so in a sentence; <code>resourceId</code> and <code>statusId</code> say which resource and which status. Status changes, attestations, archiving, connections pointing <i>at</i> it, and confirming a property stay open.	Move the resource on through its lifecycle, or make the change on a new revision.
<code>attestation_not_repeatable</code>	409	That event type is declared non-repeatable, and the resource already has one in its current status. <code>message</code> says so in a sentence; <code>resourceId</code> and <code>eventType</code> say which. A status change starts a new count.	Do not resend. If the act really happened again, the resource must move on first — or the type must be declared repeatable.
<code>transition_record_undeclared</code>	409	The move you asked for records an event when it completes, but that event type is not declared for this resource, or it needs values a move cannot supply. <code>message</code> says which; <code>eventType</code> names it. Nothing moved.	Do not retry — the lifecycle's declaration needs correcting. Tell whoever administers the account.

Blocked by the model

Code	Status	Meaning	What to do
port_max_connections	422	That port already holds as many connections as it allows. Carries <code>portId</code> and <code>maxConnections</code> .	Change the existing connection (<code>replaceInteractionId</code>) or remove one first.
port_inactive	422	That port is switched off and takes no connections.	Use another port.
accepted_type	422	That port does not accept resources of the other side's type.	Pick a resource of an accepted type (see the port's <code>effectiveTypes</code>).
port_direction	422	The two ports face the wrong way for this connection. The source port must send (<code>out</code> or <code>bidirectional</code>) and the target port must receive (<code>in</code> or <code>bidirectional</code>). Carries <code>sourcePortId</code> and <code>targetPortId</code> .	Swap the source and target, or use ports that face each other. <code>legal-targets</code> lists only ports that do.
compat_authorizing_session_only	403	Compatibility markings are authored by signed-in people, not keys.	Ask a person to mark it in the web application.
derive_target_write_blocked	400	A property in your patch is computed by a formula. <code>keys</code> names them and where each is computed.	Remove those keys. They are maintained for you and a manual value would be overwritten.
attestation_status_gate_unmet	422	That event type requires the resource to be at a specific status first. Carries <code>requiredSubjectStatusId</code> and <code>currentStatusId</code> .	Move the resource to the required status, then attest.
compat_deny	422	The membership would put an item under a setting it is marked incompatible with — a choice made on the new container (or above it) that passes down to what it contains. <code>message</code> names the item and the setting. May carry <code>viaAncestorId</code> (where the setting is made), <code>offendingResourceId</code> (the item that conflicts) and <code>offenderCount</code> ; an id is left out when your key cannot read that resource.	Change the conflicting choice, or put the item somewhere else.
contents_locked	422	The new container (or one above it) locks a setting, and the item — or something inside it — uses a different value. Same optional fields as <code>compat_deny</code> , plus <code>kindTypeId</code> .	Align the item's setting with the locked one, or put it somewhere else.
containment_cycle	422	The resource you are adding is the container itself, or already contains it — directly or further down. Adding it would put the container inside itself. <code>message</code> names both.	Pick a different resource, or a different container.

Code	Status	Meaning	What to do
containment_capacity	422	The resource's type limits how many containers of the destination's type it can be inside, and it is already inside that many (a move does not count the container it leaves). Carries <code>childId</code> , <code>parentTypeId</code> and <code>maxParents</code> ; <code>message</code> names the type.	Take it out of one of those containers first, or move it instead.
compat_type_no_t_connectable	422	A compatibility type (<code>compatible_with</code> / <code>incompatible_with</code>) was used for a connection on a port. Compatibility is set between resources, not as a connection. Carries <code>interactionTypeId</code> .	Use a type from the port's <code>allowedInteractionTypes</code> (GET <code>/v1/resources/:id/ports</code>).
capture_shape	422	A transition's <code>capture</code> is not in the declared shape: an object naming <code>properties</code> , <code>settings</code> or <code>viaPorts</code> , each as a non-empty list or "all" (<code>viaPorts</code> maps a port name to such a list). Raised by the web application's lifecycle authoring; keys never author lifecycles.	Fix the declaration.

Advisory reasons (legal-targets only)

These appear inside a `legal-targets` candidate's `reasons`, never as a response status.

Code	Status	Meaning
no_viable_target_port	—	The candidate passes every rule, but none of its ports can take this connection (direction, accepted type or its own capacity).
status_read_only	—	The connection's source — the anchor, or the candidate when the anchor's port receives — sits in a read-only status, so a create would be refused with <code>409 status_read_only</code> . <code>message</code> names it.

Refusals passed through from the create/alias doors

Code	Status	Meaning
find_or_create_refused	400 / 403 / 409	<code>find-or-create</code> refused; the reason is in <code>message</code> . <code>403</code> : the type is not in <code>find_or_create_types</code> . <code>400</code> : the external system is unknown; or it is <code>resync</code> ("find-or-create is for external systems; 'resync' is the canonical system"); or the type is a built-in system type ("Cannot mint a resource of a system type"). <code>409</code> : the identifier is already bound to a resource of another type ("... is already bound under ... to a resource of another type"), or to a resource your key cannot read; in neither case is that resource named.
alias_bind_refused	400 / 403 / 404 / 409	<code>aliases/bind</code> or <code>identity-claims/redeem</code> refused; reason in <code>message</code> . <code>403</code> : the external system is not in your key's <code>bind_systems</code> ("key not scoped to bind under '...'") — on <code>redeem</code> this is checked before the code. <code>409</code> : the identifier is already bound elsewhere; the other resource is not named. A target outside your organization is <code>403 subject_outside_grant</code> .

Device credentials — check-in, enrolment and acting for a person

A device credential ([auth-and-scopes.md](#)) meets these; an integration key meets only the first, on the device doors.

Code	Status	Meaning	What to do
<code>presence_scopes_required</code>	403	This door is for a device credential: check-in needs <code>acts_at_resource_id</code> and <code>actor_source</code> ; enrolment redeem needs <code>acts_at_resource_id</code> .	Use a device credential.
<code>acts_at_resource_missing</code>	403	The resource the device credential acts at no longer exists.	Ask your administrator for a new credential.
<code>acts_at_outside_owner_grant</code>	403	The resource the device credential acts at no longer belongs to the credential's organization.	Ask your administrator for a new credential.
<code>system_required</code>	400	Check-in without <code>system</code> .	Send the identifier's system by name.
<code>unknown_system</code>	400	No identifier system has that name.	Fix the request.
<code>alias_code_invalid</code>	400	The code is empty once the system's own rule is applied (for example, nothing but punctuation).	Read the identifier again.
<code>alias_not_enrolled</code>	404	No one is linked to that identifier.	Link it first (enrolment redeem).
<code>no_active_session</code>	404	No one is checked in at this device.	Check someone in.
<code>presence_session_required</code>	400	A write by a device credential with <code>actor_source</code> without <code>presenceSessionId</code> .	Send the id from the check-in.
<code>presence_session_expired</code>	401	The <code>presenceSessionId</code> names a session that ended, expired, or belongs to another device.	Check the person in again.
<code>actor_source_presence_session_only</code>	403	A device credential with <code>actor_source</code> named an actor (<code>actorResourceId</code> / <code>actorAlias</code>).	Leave them out; send <code>presenceSessionId</code> .
<code>enrollment_code_required</code>	400	Redeem without an 8-character <code>enrollmentCode</code> .	Send the code the administrator issued.
<code>enrollment_not_found</code>	404	No pending enrolment has that code — it is wrong, already used, or cancelled.	Ask for a new code.
<code>enrollment_expired</code>	410	The enrolment code has expired.	Ask for a new code.
<code>enrollment_outside_owner_grant</code>	403	The enrolment belongs to another organization than the device credential's.	Redeem it on a device of that organization.
<code>alias_already_bound</code>	409	The identifier is already linked to someone else.	Have it unlinked first.

Connected apps

An app registered to sign people in ([sign-in.md](#)). A connected app is an app registered to sign people in with ReSync: its redirect URIs and the scopes it may use for a person. An administrator of the owning organization registers and manages it from the organization's page in ReSync; those administration endpoints are for a signed-in person only and are not part of this API. A key meets only the first code below. A refusal of the app's `allowed_scopes` reads as the scope-validation refusal of key administration.

Code	Status	Meaning	What to do
connected_app_managed	403	The resource is a connected app. Its name, redirect URIs and scopes change only through its registration, never through <code>PATCH /v1/resources/:id</code> ; and nothing is added to it or removed from it as a member (enrol, exit or move) — its machine keys are managed with the app.	Ask an administrator of the owning organization to change the registration.
invalid_client_code	400	The app's code is not 2–63 lowercase letters, digits and hyphens, starting with a letter or digit.	Choose another code.
client_code_taken	409	Another connected app already has that code. Codes are never reused.	Choose another code.
client_code_immutable	400	An app keeps its code for life.	Register a new app for a new code.
invalid_client_name	400	The name is empty or longer than 200 characters.	Fix the name.
invalid_client_kind	400	<code>kind</code> must be <code>public</code> .	Send <code>public</code> .
invalid_client_owner	400	A connected app is registered by an organization, not a person.	Register it under the organization.
invalid_redirect_uri	400	A redirect URI is not an absolute <code>https</code> URL, carries a fragment or credentials, or there are more than 20 — or it names a loopback host (<code>localhost</code> , <code>127.0.0.1</code> , <code>:::1</code> , <code>*.localhost</code>), which the production issuer refuses whatever the scheme. <code>message</code> names the URI.	Fix the URI. Redirect URIs are matched exactly.
connected_app_not_found	404	No connected app has that client id.	Check the id.
connected_app_revoked	409	The app is revoked. A revoked app is never changed or reactivated.	Register a new app.
invalid_client_transition	409	The app cannot move to that status from where it is (only active $\Rightarrow$ suspended, and either $\rightarrow$ revoked), or it changed meanwhile.	Read the app again.
key_owner_mismatch	409	The key belongs to a different organization than the app.	Use a key of the app's organization.
invalid_client_status	400	The requested status is not one of <code>active</code> , <code>suspended</code> or <code>revoked</code> .	Choose one of the three.

Code	Status	Meaning	What to do
<code>key_scope_exceeds_app</code>	409	The key may use types the app may not: its operate, attest or read lists reach past the app's allowed types. <code>message</code> names those types.	Narrow the key, or widen the app's allowed types first.
<code>key_attached_elsewhere</code>	409	The key already belongs to another connected app. A machine key serves one app.	Mint a new key for this app.
<code>key_not_attached</code>	409	That key does not belong to this app (it may have been rotated or revoked already).	Use one of the app's current keys.

Sign in with ReSync

How an app signs a person in: [sign-in.md](#). Signing a person in to an app is done by the person, in ReSync's web app. A key meets only `session_required`. The token and userinfo endpoints of `https://auth.limnal.dev` answer in the standard OAuth 2.0 / OpenID Connect error shape instead ({ "error", "error_description" }).

Code	Status	Meaning	What to do
<code>session_required</code>	403	Signing in to an app is done by a signed-in person, not a key.	Sign the person in through the app.
<code>invalid_client</code>	400	<code>client_id</code> is not a client id.	Use the client id your app was given.
<code>redirect_uri_mismatch</code>	400	<code>redirect_uri</code> is not one of the app's registered redirect URIs. They are matched exactly.	Register the URI, or send one that is registered.
<code>unsupported_scope</code>	400	<code>scope</code> must include <code>openid</code> and may add <code>profile</code> — nothing else.	Ask for <code>openid</code> (and <code>profile</code> for the name).
<code>invalid_code_challenge</code>	400	The PKCE challenge is missing, is not 43–128 base64url characters, or its method is not <code>S256</code> .	Send an S256 challenge of your code verifier.
<code>invalid_state</code>	400	<code>state</code> is missing or longer than 512 characters.	Send a state of 1–512 characters.
<code>invalid_nonce</code>	400	<code>nonce</code> is longer than 255 characters.	Send a shorter nonce, or none.
<code>client_not_active</code>	401	The access token is for an app that is suspended or revoked.	Ask the app's owner; sign in again once it is active.
<code>client_not_permitted</code>	403	An app signed in for a person cannot use this endpoint (messages, keys, access administration, the person's own settings and others).	Call it as the person in ReSync's web app, or not at all.

Live updates

Only for those who may hold the stream (signed-in people and device credentials).

Code	Status	Meaning	What to do
<code>live_unavailable</code>	503	The live stream is not available right now.	Retry later; meanwhile read the resource.
<code>live_connection_limit</code>	429	Ten streams are already open for this caller.	Close one first.
<code>live_connection_not_found</code>	404	The <code>connectionId</code> is not an open stream of yours (it closed, or the server restarted).	Open a new stream.
<code>live_view_limit</code>	422	A stream can follow at most 50 views.	Follow fewer.
<code>invalid_views</code>	400	<code>views</code> is not a list of <code>{ resourceId, view }</code> with <code>view</code> one of <code>wall</code> , <code>members</code> , <code>inbox</code> , <code>canvas</code> , <code>lifecycle</code> .	Fix the request.

Server — not your request

Code	Status	Meaning	What to do
<code>internal_error</code>	500	Something went wrong on our side. The body is always <code>{ "error": "internal_error", "message": "Something went wrong.", "requestId": "..." }</code> — never the cause.	Quote <code>requestId</code> (it is also the <code>x-request-id</code> header) when you report it. Retry once with backoff, as Retrying describes.

Transport

Code	Status	Meaning	What to do
<code>rate_limit_exceeded</code>	429	Your key used its whole budget for the current one-minute window (300 by default; <code>x-ratelimit-limit</code> says your key's own). The window is fixed: it started with your first request. Carries <code>retryAfterSeconds</code> and a <code>retry-after</code> header — the seconds left in the window.	Wait the stated time, then continue.

Retrying

Only **429** is safe to retry blindly. Wait for `retry-after`.

5xx (`internal_error`, `live_unavailable`) may be retried once with backoff, but read [conventions.md](#) first: writes are not idempotent, so a `PATCH` that timed out may already have been applied. Read the resource back before resending it.

Never retry a `401`, `403`, `404` or `409` unchanged. Every one of them means the request will be refused the same way until something changes on your side or your administrator's.

8 Events

Every change in ReSync appends an event. Nothing is overwritten, so a resource's history is complete and permanent — including the changes you make.

Reading history

```
curl -s -H "Authorization: Bearer $TOKEN" "$BASE/v1/resources/<bidId>/events?limit=3"
```

Response: { items: [...], nextCursor: string | null, afterCursor: string }, newest first. Read scope for the resource's type is required, and the resource must belong to your organization:

From `events-outside-grant.json` — status 403:

```
{
  "error": "subject_outside_grant"
}
```

Parameter	Type	Default	Notes
<code>limit</code>	string	30	Clamped to 1–100.
<code>cursor</code>	string	—	From a previous <code>nextCursor</code> .
<code>filter</code>	string	<code>all</code>	See below.
<code>after</code>	string	—	Poll for what changed since you last looked — see Polling for changes .

Filters

- `all` — the default. Messages and the curated set of notable changes.
- `changes` — the full change stream, including low-level entries the default view leaves out. Use this when you are reconciling data rather than showing a human what happened.
- `messages` — conversation only.
- `quiet` — configuration entries only: a lifecycle, a consequence rule, a view's settings or a status's attention changed. They are recorded on the thing configured — and only there: a configuration change never appears on the history of a resource that merely uses or contains the thing configured. They are left out of `all` and `changes`; ask for them here.

Anything else in `filter` is treated as `all` rather than rejected.

Polling for changes

To keep up with a resource, read its history once, keep the `afterCursor` from that response, and from then on ask only for what came after it:

```
curl -s -H "Authorization: Bearer $TOKEN" "$BASE/v1/resources/<bidId>/events?
filter=changes&after=$AFTER"
```

With `after`, events come **oldest first**, and `nextCursor` is **always** present — it is where your next poll starts, even when nothing new has happened (then it is the cursor you sent). Store it and send it next time. `after=start` begins at the very first event.

From `events-after-page.json` — status 200:

```
{
  "items": [
    {
      "event_ulid": "01M5FWM4P0A96QHDS7NPX9H7RE",
      "event_type": "resource.created",
      "occurred_at": "2026-10-21T18:47:21.024Z",
      "actor": {
        "resource_id": 24907,
        "name": "bid-app-docs"
      },
      "display": {
        "name": "Harbor Homes – kitchen",
        "resource_type_id": 176,
        "text": "* created – Bid"
      },
      "producer_id": "bid-app-docs"
    },
    {
      "event_ulid": "01M5FWM4P38KF0C0EDHS6AQKRS",
      "event_type": "status.changed",
      "occurred_at": "2026-10-21T18:47:21.028Z",
      "actor": {
        "resource_id": 24907,
        "name": "bid-app-docs"
      },
      "display": {
        "from_status_id": null,
        "to_status_id": 280,
        "from_status_name": null,
        "to_status_name": "Draft",
        "text": "set to Draft"
      },
      "producer_id": "bid-app-docs"
    },
    {
      "event_ulid": "01M5FWM4QB2SJNA1F1TMY6R5R5",
      "event_type": "property.changed",
      "occurred_at": "2026-10-21T18:47:21.068Z",
      "actor": {
        "resource_id": 24907,
        "name": "bid-app-docs"
      },
      "display": {
        "key": "tax_rate_bp",
        "from": null,
        "to": 825,
        "text": "🔗 tax_rate_bp: – → 825"
      },
      "producer_id": "bid-app-docs"
    }
  ],
  "nextCursor": "c1MTMwMTI1MS4yMzY3OQ"
}
```

It never skips an event. Several writes can be in progress at once, and one that started earlier can finish later. A poll only hands out events whose writes have fully finished *and* that nothing still in progress could land in front of, so an event cannot appear behind a cursor you already hold. The cost is that a just-finished write can take a moment longer to show up — the next poll has it.

Your first poll may repeat a few events you already saw on page one (at most the ones written while you were reading). Treat events as keyed by `event_ulid` and drop ones you have.

A cursor is opaque: store it, send it back, never build or edit one. Anything else answers `400` :

From `events-after-invalid.json` — status `400`:

```
{
  "error": "invalid_cursor",
  "message": "Invalid cursor"
}
```

One known gap. An event sent through event ingest whose resource could not be identified when it arrived is attached to that resource later. It then appears in the newest-first history, but it is not handed out to `after` polls, because it was written before your cursor. If your integration sends events it cannot yet resolve, re-read page one occasionally.

What your writes look like

Three fields say who did something, and they answer three different questions. They are worth understanding once, because they are how your integration is accountable later.

Field	Answers	For your writes
<code>producer_id</code>	Which system wrote this?	Your key's name — on every change it makes, including the birth of a resource it creates (<code>resource.created</code> and the first <code>status.changed</code>) and of a person it names by <code>actorAlias</code> for the first time. A resource ReSync creates in response to your write — the order a converted bid produces — carries the name of the ReSync process that created it (<code>consequence.worker</code>); a status ReSync advances on its own carries <code>advance.worker</code> . Another integration's key name is shown only when you can read its service identity; otherwise <code>someone</code> .
<code>actor</code>	On whose behalf?	A service identity that belongs to your key, named after it. An actor you cannot read keeps its <code>resource_id</code> , reads <code>name: "someone"</code> and carries <code>readable: false</code> .
<code>provenance</code>	How is this known?	<code>attested</code> — somebody or something asserted it, as opposed to sensed or inferred.

So a change made by your integration is distinguishable from one made by a person in the web app, and from one computed by a formula, without anybody having to annotate it.

Naming a different actor. If your system knows *which person* did the thing, say so — pass `actorResourceId` (a person's resource id) or `actorAlias` (`{ system, code }` , an identity in your own system) on the write. The producer stays your key either way: the key is still the system that wrote it. Omit both and the service identity is used, which is honest when the action really was the software's.

Who may be named. `actorResourceId` may name any person your key can read — in practice, any person of your organization. That is how an approval is attributed to the person who approved it (for example an internal acceptance recorded as `bid.accepted_internally`). Naming someone outside your organization is refused, and nothing is recorded:

From `attestation-403-actor-not-readable.json` — status 403:

```
{
  "error": "actor_not_readable"
}
```

Event types you will meet

Event type	When
<code>property.changed</code>	One property changed. Carries <code>key</code> , <code>from</code> , <code>to</code> — one event per key.
<code>resource.renamed</code>	The name changed. Carries <code>from_name</code> , <code>to_name</code> .
<code>resource.reidentified</code>	The identifier changed. Carries <code>from_identifier</code> , <code>to_identifier</code> .
<code>property.unconfirmed</code>	A confirmed value was superseded by an edit. Your key cannot cause this — it is refused instead.
<code>status.changed</code>	The resource moved to a new status.
<code>component.enrolled</code>	A resource was put inside a container. A move's enrolment also carries <code>moved_from_resource_id</code> .
<code>component.exited</code>	A resource left a container. A move's exit also carries <code>moved_to_resource_id</code> ; both halves share one transaction.
<code>resource.created</code>	The resource came into existence. When it was created with properties, carries <code>properties</code> (the values it was born with — no separate <code>property.changed</code> follows); when it was created by <code>find-or-create</code> , carries <code>alias: { system, code }</code> .
<code>resource.archived / resource.unarchived</code>	Archival state changed.
<code>resource.tagged / resource.untagged</code>	A tag was applied to or removed from the resource. Carries <code>tagId</code> and <code>tagName</code> (the tag's name when it happened).
<code>resource.owner_changed</code>	The resource's owner changed. Carries <code>fromOwnerId</code> and <code>toOwnerId</code> .
<code>port.created / port.updated / port.deleted</code>	A port on the resource was added, changed or removed by hand. Carries <code>resource_port_id</code> , <code>port_name</code> , and on an update <code>changedKeys</code> (plus <code>added_resource_type_id</code> or <code>removed_resource_type_id</code> when the accepted types changed). In the <code>changes</code> filter only.
<code>port_definition.changed</code>	A type's port definition changed its own settings. Recorded on the type's resource; carries <code>definitionId</code> , <code>port_name</code> and <code>changedKeys</code> . In the <code>changes</code> filter only.
<code>resource_type.prof</code>	A type's card presentation changed — its icon, its key properties, or both. Recorded on the type's resource only (never on a container's history); carries <code>typeId</code> and <code>changedKeys</code> (<code>icon</code> , <code>keyProperties</code>), never the values. In the

Event type	When
file_changed	all and changes filters; never unread.
resource_type.config_changed	A type's configuration changed. Today: an emission was declared or removed (kind: "emission", verb created / deleted). Recorded on the type's resource only; carries kind, verb, id, typeId, attestationEventTypeId and eventTypeKey (the event type's code). In the all and changes filters; never unread.
resource.captured	A status change fired a transition that declares a capture, and the resource was photographed at that moment — one event every time the transition fires. display carries transitionId, fromStatusId, toStatusId, properties (the resource's own values the transition named) and, when something could not be recorded because the resource's owner cannot read it, unreadable (the setting codes or port names left out); the sentence says how many values were recorded. The photographed settings and attached resources are kept on the record but are not passed into display, since they name other resources. In the all and changes filters; never unread.
thread.created	A conversation was started. Recorded on the conversation (subject_key thread:{id}) and, when it belongs to a resource, scoped to that resource — so it can appear in that resource's events. Carries thread_id, creator_resource_id, context_resource_id. In the all and changes filters; never unread.
thread.participant_added / thread.participant_left	A person joined or left a conversation (self: true when they left on their own). Same placement as thread.created; carries thread_id and participant_resource_id.
attestation.refused	An event your producer sent through event ingest was not recorded: its type is declared, and the resource was in the wrong status for it, or it is not repeatable and was already recorded in the current status. Carries eventType, code (attestation_status_gate_unmet or attestation_not_repeatable) and producerEventId. It takes the place of the refused event, under your producer id, so resending the same event is recognized as a duplicate rather than refused twice.
lifecycle.config_changed	A lifecycle's configuration changed — the lifecycle itself, one of its statuses, a transition or a requirement. Recorded on the lifecycle's own history only (never on a resource that runs it, nor on a container); carries kind, verb, id, changed, before / after of the changed fields and the ids it touched (statusId, transitionId, requirementIds ...) — typed words such as names and labels are noted as changed, not copied. In the quiet filter only.
consequence_rule.changed	A consequence rule (an automatic follow-on, such as the order a converted bid produces) was created, changed or retired. Recorded on the rule's own history only; carries verb, id, changed, before / after and the rule's ids (triggerEventType, subjectTypeId, mintTargetTypeId ...). In the quiet filter only.
derivation.refused	A computed value or a templated name was not updated. Carries key and code — unique_key_taken, status_read_only, derive_target_not_derivable, or setting_name_unreadable (the name would use a setting value the resource's owner cannot read). In the changes filter.
presence.checked_in	A person checked in at a device credential's resource. Recorded on that resource, with the person as the actor and the credential's name as the producer; carries presence_session_id, api_key_id and via_system. Never unread.
presence.checked_out	A check-in ended. Same placement; carries presence_session_id and reason: tap_out (the device checked the person out), superseded (someone else checked in), expired (no activity in time) or revoked (the credential was revoked, or the identifier unlinked).
identity.alias_enr	An identifier was linked to a person through enrolment redeem. Recorded on the person; the actor is the administrator who issued the code, the producer the device credential that redeemed it. Carries system and code.

Event type	When
olled	
credential.connect.ed	A device credential was first used from a new device (<code>x-device-id</code> , auth-and-scopes.md). Recorded on the credential's resource, with the credential's name as the producer; carries <code>api_key_id</code> , <code>device_id</code> and <code>user_agent</code> . In the <code>all</code> and <code>changes</code> filters; never unread.
connected_app.registered	An app was registered to sign people in (sign-in.md). Recorded on the organization that owns it; carries <code>connected_app_resource_id</code> , <code>code</code> , <code>kind</code> , <code>redirect_uris</code> and <code>allowed_scopes</code> — the details for that organization's administrators only.
connected_app.updated	An app's registration changed. Same placement; carries <code>connected_app_resource_id</code> , <code>changed</code> (the fields) and their <code>before</code> / <code>after</code> — administrators only.
connected_app.signed_in	A person signed in to an app (one row per sign-in). Recorded on the app, with the person as the actor; carries <code>person_resource_id</code> , <code>scope</code> and <code>jti</code> — administrators only.
your attestation types	Whatever <code>POST /v1/resources/:id/attestations</code> recorded — the codes your organization has declared. Read them from <code>GET /v1/attestation-event-types</code> . A declaration can change, so read it rather than caching it — since 2026-10-17 <code>bid.sent</code> declares the six totals and <code>tax_rate_bp</code> only (<code>customer_pct_bp</code> was removed; the price level and payment terms are now recorded from the bid's own fields when it is sent).

A sent bid's newest history, as a key reads it — the status change, the snapshot Submit took, and `bid.sent` with the current declaration's fields:

From `events-grant-owned.json` — status 200:

```

{
  "items": [
    {
      "event_ulid": "01M5FWMHNVHG3M0ZEG8ZNM1HF5",
      "event_type": "status.changed",
      "occurred_at": "2026-10-21T18:47:34.331Z",
      "actor": {
        "resource_id": 24907,
        "name": "bid-app-docs"
      },
      "display": {
        "from_status_id": 280,
        "to_status_id": 1314,
        "from_status_name": "Draft",
        "to_status_name": "Sent",
        "text": "Draft → Sent"
      },
      "producer_id": "bid-app-docs"
    },
    {
      "event_ulid": "01M5FWMHNTJHEV5ZTHPXZ9NADP",
      "event_type": "resource.captured",
      "occurred_at": "2026-10-21T18:47:34.330Z",
      "actor": {
        "resource_id": 24907,
        "name": "bid-app-docs"
      },
      "display": {
        "transitionId": 818,
        "fromStatusId": 280,
        "toStatusId": 1314,
        "properties": {
          "tax_rate_bp": 825
        },
        "text": "Recorded 12 values on Submit"
      },
      "producer_id": "bid-app-docs"
    },
    {
      "event_ulid": "01M5FWMHJY4Q8TB2KSHCS8Z9S9",
      "event_type": "bid.sent",
      "occurred_at": "2026-10-21T18:47:34.238Z",
      "actor": {
        "resource_id": 24907,
        "name": "bid-app-docs"
      },
      "display": {
        "tax_cents": 12790,
        "tax_rate_bp": 825,
        "total_cents": 167817,
        "install_cents": 50000,
        "delivery_cents": 0,
        "discount_cents": 0,
        "subtotal_cents": 155027,
        "attested_action_label": "Send bid"
      },
      "producer_id": "bid-app-docs"
    }
  ],
  "nextCursor": "01M5FWMHJY4Q8TB2KSHCS8Z9S9",
  "afterCursor": "c1MTMwMTMxMS4yMzc2Mg"
}

```

Each item also carries a prepared display sentence, so you can show history to a human without writing a formatter per event type. The sentence and every name in `display` follow the [names you cannot read](#) rule: something you cannot read is `someone`, `a resource` or `a type`, with its id kept and `readable: false` (`sourceReadable` / `targetReadable` on a connection event). For the event types the platform writes, `display` carries a fixed set of keys; any other key in the stored event is left out.

Two things to know

One `PATCH` can produce several events. Change three properties and rename a resource in one request and you get four events, not one. They share a timestamp and appear together.

A refused write produces none. When a request is rejected — a confirmed property, a computed property, a read-only status, a scope refusal — nothing is appended. There is no record of the attempt, so if you need to know that your integration tried and failed, log it on your side. The one exception is event ingest, which answers `202` before the event is processed: a declared event refused there is recorded as `attestation.refused` (above), so the refusal is visible on the resource's history. Ingest checks only the event's status and repeat rules; the payload-field checks of `POST /v1/resources/:id/attestations` apply to that door only. An event whose subject cannot be resolved when it arrives is recorded unchecked.

9 Conventions

The rules that hold everywhere, and the limits worth knowing before you design around them.

Shapes

Ids are strings. `resourceId`, `resourceTypeId` and similar come back as strings because some exceed what a JavaScript number holds exactly. Keep them as strings; do not parse them to compare.

Some nested fields are numbers rather than strings — `ownerResourceId` in a list row, for instance. This is an inconsistency, not a signal. Compare ids as strings after coercing, and it will not bite you.

`ownerOrgId` on a single-resource read is an internal identifier and may change; use `ownerResourceId`, which is the owning resource's id.

Timestamps are ISO-8601 in UTC, e.g. `2026-09-14T02:00:12.772Z`. Always UTC; there is no local time anywhere in the API.

Fields are camelCase, with two exceptions that predate the rest of the surface and are called out where they appear: `POST /v1/resources/find-or-create` and `POST /v1/aliases/bind` take `snake_case` bodies.

Cursors are opaque. Pass back exactly what you were given.

Errors have one shape — with one exception. Every endpoint answers a refusal as `{ "error": "<code>", "message": "..." }` ([errors.md](#)), except the two OAuth endpoints of sign-in (`/oauth/token`, `/oauth/userinfo`), which use the standard OAuth 2.0 / OpenID Connect shape `{ "error", "error_description" }` because client libraries expect it ([sign-in.md](#)).

Unknown query parameters are ignored, not rejected — a misspelt parameter does nothing and the request answers as if it were absent. Check names against [endpoints.md](#). (A parameter a door knows but cannot apply to a key, such as `sort` on the resource list, is refused with a code.)

Paging

`limit` is clamped to 1–100 everywhere. Defaults differ: the resource list defaults to **50**, the events read to **30**. Always send an explicit `limit` and the difference stops mattering.

Writing properties

properties merges shallowly. Keys you do not send are left alone. Nested objects are replaced wholesale, not merged — send the complete nested value.

You cannot delete a property key. Sending `{"colour": null}` stores a null; it does not remove `colour`. There is currently no way through this API to make a key not exist again.

Moves that record an event

Some status moves record an event when they complete — converting a bid records `bid.converted`, for instance. The advancement read tells you which: each move's `records` names the event, or is `null` (endpoints). Do not post that event yourself after the move; it is already recorded, and a second one is refused.

Properties you cannot write

Two kinds of property refuse your writes, and both refuse the **entire request** rather than applying part of it. A patch either lands completely or not at all.

Confirmed properties — a person has vouched for the value. `409 confirmed_key_locked`, with `keys` naming which. A person must re-confirm before it can change; your key cannot overturn somebody's assertion. Read `confirmedKeys` on a resource, or add `include=properties` to a list, to see which are locked before you try.

Derived properties — computed by a formula from other data. `400 derive_target_write_blocked`, with `keys` naming them and where each is computed. A manual value would be overwritten the next time the formula runs, so the write is refused instead of silently lost. `derivedKeys` on a resource read tells you which they are.

If one property is both, you get the derived refusal.

Names you cannot read

A response never tells you the name of something your key cannot read. Where a field would name it, you get a neutral word instead, and the field keeps its type — a `name` is still a string:

What the field names	The word you get
a person, an organization, an actor, a counterparty	someone
any other resource	a resource
a resource type	a type
something above the resource (a parent, a setting's source)	an enclosing item
something inside the resource	an item inside
a setting's value	a setting

Ids are not secret. The id beside a hidden name is still there, so you can tell two hidden things apart, and a row that names something you cannot read carries `readable: false`. You can always rely on that flag; never compare a name against one of the words above.

What this means in practice:

- **Connections and provenance.** An `edges` row or a `derivedFrom` row to a resource you cannot read stays in the list with `readable: false`; a `derivatives` row you cannot read is left out.
- **Computed properties.** A `derivedKeys` entry whose formula lives on a parent you cannot read names it `an enclosing item` and keeps `parentResourceId`, with `readable: false`.

- **Types.** A resource whose type you cannot read reports `resourceTypeName: "a type"` and no schema.
- **Requirements.** A requirement whose condition names something you cannot read describes it with the words above.
- **Refusal messages** name only what you can read. Messages may be reworded at any time ([errors](#)); key on `error`, never on the message.

Reading efficiently

List rows omit `properties` by default. Add `include=properties` and each row gains `properties` and `confirmedKeys` — one request instead of one per row. Do that rather than listing ids and fetching each one; the difference is large over a few hundred rows.

Reference data (types, statuses, lifecycles, units) changes rarely. Fetch it at startup and keep it.

What you can't do yet

Said plainly, so you can design around it rather than discover it.

Writes are not idempotent. There is no `Idempotency-Key`. A retried `PATCH` writes its events a second time, and a request that timed out may have succeeded. **Read the resource back before resending a write.** The one exception is `POST /v1/resources/find-or-create`, which is idempotent by your own identifier — for anything you create, prefer it. A request that creates a resource together with what it holds replays the same way: each resource in it is found again by its own identifier, and nothing is created, put inside or connected twice.

You cannot delete a property key. See above.

Some surfaces are not open to keys. A resource's canvas placement, its explanation (`/why`), the compatibility candidates and every authoring surface (types, lifecycles, compatibility markings, retyping a connection) require a signed-in person; `401` or `403` there is the current design, not a misconfiguration. A key reaches a resource's **members, containers, events, advancement, requirements** and **ports** (with legal targets), the cross-type **search**, and it can enrol, exit and **move** members and make or remove an **attachment** — see [endpoints.md](#) for every door.

There is no bulk read of compatibility rules. A type's compatibility pairings (`GET /v1/resource-types/:id/compat-pairings`, and one pairing at `.../compat-pairings/:partnerTypeId`) are served to signed-in people only; a key gets `403 compat_authoring_session_only` there. A key learns what fits one field at a time: `GET /v1/resources/:id/ports/:portId/legal-targets` gives each candidate's verdict and the reasons — the same answer the attachment would get.

A key cannot link a revision to the bid it replaces. A revision is a new bid; the link from it to the one it replaces (`derives_from`) has no ports, so a key's `POST /v1/resource-interactions` is refused `400 ports_required` ([walkthrough](#)).

You cannot manage your own key. Creating, rescoping and revoking are administrator actions through the web application. Your key cannot even read the list of keys — only `GET /v1/api-keys/self`.

Keys do not expire. A key stays valid until somebody revokes it. Plan your own rotation — a connected app's machine key is rotated by your organization's administrator from the app's page (a new name, the old key revoked).

Authentication failures come back in two different shapes. On most endpoints a bad or revoked key returns `{ statusCode, error, message }` with a phrase in `error`, not a code. Treat any `401` as "check your key" — see [errors.md](#).

A few doors return sentences rather than codes for some of their own validation — attestations and status changes among them. [errors.md](#) lists every one. Match on the HTTP status there.

There is no webhook or push for keys. To notice a change you poll a resource's events with `after` and the cursor from your last poll ([events.md](#)). ReSync's own web app and device credentials receive live updates over a stream; that stream is not open to integration keys.

One write creates a resource and what it holds; everything else is one resource per request. `POST /v1/resources/find-or-create` takes the resource's fields and the resources inside it, to any depth (up to 250 in one request), and writes them all or none ([endpoints](#)) — a whole bid is one request. Changing what already exists (a `PATCH`, a status move, replacing a field) is still one request per change ([walkthrough](#)).

10 Changelog

How v1 changes

Additive by default. New endpoints, new optional parameters and new response fields can appear at any time. Your client must ignore fields it does not recognise.

Error codes are stable strings. A code that appears in [errors.md](#) keeps its meaning for the life of v1. New codes may be added; existing ones are not renamed or repurposed.

Fields are never renamed within v1. If a field's name is wrong, a new one appears alongside it and the old one stays.

Removals are announced. Anything withdrawn gets an entry here first, with a notice period before it stops working. Nothing disappears from this pack without an entry below.

Breaking changes mean v2. A new major version would be served alongside v1, not in place of it.

Regenerate `resync-api-v1.pdf` whenever an entry is added here.

v1 · 2026-10-28 — wording

Wording only; no door, field, scope or behaviour changed.

- [auth-and-scopes.md](#) — "The two scope sets" is now "The scope sets": the three type sets in the table, `find_or_create_types` for find-or-create, and `bind_systems` (external systems, not types).
 - [endpoints.md](#) — `POST /v1/resources/:id/attestations` and `PATCH /v1/resources/:id/status` say "Needs attest or operate scope", matching the scope page.
 - [README.md](#) — ownership notes that a person can own a resource where personal modeling is on.
-

v1 · 2026-10-26 — initial surface

The first entry of v1: the surface below is what this pack documents, and every rule above applies from here on.

- **Resources** — list, search, read, change and contain: `GET /v1/resources`, `GET /v1/resources/search`, `GET /v1/resources/:id`, `PATCH /v1/resources/:id`, `PATCH /v1/resources/:id/status`, `POST /v1/resources/find-or-create`, `GET /v1/resources/:id/members`, `POST /v1/resources/:id/members`, `POST /v1/resources/:id/exit`, `POST /v1/resources/:id/move`, `GET /v1/resources/:id/containers` ([endpoints.md](#), [walkthrough-bid.md](#)).
- **History and requirements** — `GET /v1/resources/:id/events`, `GET /v1/resources/:id/advancement`, `GET /v1/resources/:id/capabilities`, `GET /v1/resources/:id/requirements`, `POST /v1/resources/:id/attestations`.

- **Ports and connections** — GET `/v1/resources/:id/ports`, GET `/v1/resources/:id/ports/:portId/legal-targets`, POST `/v1/resource-interactions`, DELETE `/v1/resource-interactions/:id`.
- **The model, read-only** — GET `/v1/resource-types`, GET `/v1/resource-types/:id`, GET `/v1/lifecycles`, GET `/v1/lifecycles/:id`, GET `/v1/lifecycles/:id/graph`, GET `/v1/state-transitions`, GET `/v1/statuses`, GET `/v1/attestation-event-types`, GET `/v1/property-keys`, GET `/v1/units`, GET `/v1/interaction-types`.
- **Aliases and identity claims** — POST `/v1/aliases/bind`, GET `/v1/aliases/resolve`, POST `/v1/identity-claims/redeem`.
- **Credentials** — a key reads itself and its default view, GET `/v1/api-keys/self`, GET `/v1/api-keys/self/view`; scopes, the per-key rate limit and request ids ([auth-and-scopes.md](#)).
- **Device credentials and the station plane** — POST `/v1/presence-sessions`, GET `/v1/presence-sessions/current`, POST `/v1/presence-sessions/current/touch`, DELETE `/v1/presence-sessions/current`, POST `/v1/alias-enrollments/redeem`; the live-update stream GET `/v1/live` ([auth-and-scopes.md](#)).
- **Sign in with ReSync** — OpenID Connect, authorization code + PKCE, issuer `https://auth.limnal.dev`: GET `/.well-known/openid-configuration`, GET `/.well-known/jwks.json`, POST `/oauth/token`, GET `/oauth/userinfo`; the consent screen's GET `/v1/oauth/clients/:id` and POST `/v1/oauth/authorize` take a person's session only ([sign-in.md](#)).
- **Errors, events, conventions** — [errors.md](#), [events.md](#), [conventions.md](#).