

ReSync API v1

Developer Guide

2026-10-20

generated from `docs/api/v1`

Contents

1. Introduction

2. Quickstart

3. Authentication and scopes

4. Endpoints

5. Errors

6. Events

7. Conventions

8. 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. 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.

Next: [quickstart.md](#) — one whole bid, from an empty start to a converted order, with a real response for every door.

2 Quickstart

One whole bid, from an empty start to a converted order, on an integration key. Every response below is a real capture from a running server — nothing here is illustrative. The resources in the captures are a test customer, "Harbor Homes", and its kitchen bid; your ids, names and price levels will differ.

The walk is the order an integration actually needs the doors in:

#	Step	Doors
1	Who am I	GET /v1/api-keys/self
2	Map type codes to ids; read the limits a type declares	GET /v1/resource-types , GET /v1/resource-types/:id
3	Read the vocabulary you may attest	GET /v1/attestation-event-types
4	Read reference data (price levels, payment terms)	GET /v1/resources?typeKey=...&include=properties
5	Bring in the customer by your own identifier	POST /v1/resources/find-or-create
6	Set its properties	PATCH /v1/resources/:id
7	Fill its fields (price level, payment terms)	GET .../ports , GET .../ports/:portId/legal-targets , POST /v1/resource-interactions
8	The bid, its properties and its three fields	the same doors
9	A design set inside the bid	POST /v1/resources/find-or-create , POST /v1/resources/:id/members
10	Lines inside the set, each with its product	find-or-create, members, PATCH , legal-targets with q , POST /v1/resource-interactions
11	Read the tree back	GET /v1/resources/:id/members , GET /v1/resources/:id
12	Preview what a status move needs	GET /v1/state-transitions , GET /v1/resources/:id/requirements
13	Freeze bottom-up: lines, then the set	PATCH /v1/resources/:id/status
14	Submit: record the totals, then send	POST /v1/resources/:id/attestations , PATCH .../status
15	Accept and convert; find the order	attestations, PATCH .../status , GET /v1/resources/:id

Steps that use the same doors again (a second field, the next line) are shown once. examples.http has one runnable block per door, in this order. The latest copy of this walk 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": 781,
  "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. Read the list once and keep the `code → id` map:

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

The full answer is `resource-types-list.json` in the fixture set — one row per type your key can read, each with its `code` and `resourceTypeId`. 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/3425"
```

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 you may attest

An **attestation** records that something happened — a bid was sent, accepted, converted — without changing the resource's own fields. The event types and the fields each declares are reference data:

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

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

4. Read reference data

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. Bring in the customer by your own identifier

`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.

```

curl -s -X POST -H "Authorization: Bearer $TOKEN" -H 'Content-Type: application/json' \
  -d '{"type_key": "cab_customer", "external_system": "pdmac-sql", "external_key": "cust-10442", "name": "Harbor Homes", "properties": {"kind": "contractor"}}' \
  "$BASE/v1/resources/find-or-create"

```

From `foc-create.json` — status 201:

```
{
  "resource_id": 24786,
  "created": true
}
```

Called again with the same identifier:

From `foc-find.json` — status 200:

```
{
  "resource_id": 24786,
  "created": false
}
```

What to notice. The second call does not update anything — see [Things to know](#). `external_system` names an external system ReSync already knows; your administrator tells you which.

6. Set its properties

```
curl -s -X PATCH -H "Authorization: Bearer $TOKEN" -H 'Content-Type: application/json' \
  -d '{"properties":{"customer_note":"call before delivery"}}' "$BASE/v1/resources/$CUSTOMER"
```

`properties` **merges** — keys you do not send are left alone. The answer is the resource as it now stands (a capture is under `PATCH /v1/resources/:id` in [endpoints.md](#)).

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

7. Fill its fields

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

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

A picker over a long list narrows with `q` and pages with `limit` (here a line's `product` field over the price book):

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

```

{
  "sourcePort": {
    "portId": 6707,
    "portName": "product",
    "direction": "out",
    "maxConnections": 1,
    "held": 0,
    "capRemaining": 1
  },
  "candidates": [
    {
      "resourceId": 24699,
      "name": "WD Laminate Modern Vertical 30",
      "resourceTypeId": 191,
      "verdict": "legal",
      "reasons": [],
      "viableTargetPorts": [
        {
          "portId": 6623,
          "portName": "Sold as",
          "direction": "in"
        }
      ]
    },
    {
      "resourceId": 24697,
      "name": "WD Laminate Modern Vertical",
      "resourceTypeId": 191,
      "verdict": "legal",
      "reasons": [],
      "viableTargetPorts": [
        {
          "portId": 6622,
          "portName": "Sold as",
          "direction": "in"
        }
      ]
    },
    {
      "resourceId": 13812,
      "name": "MSDEV Line 2 W30-1/2*12",
      "resourceTypeId": 182,
      "verdict": "legal",
      "reasons": [],
      "viableTargetPorts": [
        {
          "portId": 6621,
          "portName": "Sold as",
          "direction": "in"
        }
      ]
    },
    {
      "resourceId": 12934,
      "name": "WOCM3",
      "resourceTypeId": 187,
      "verdict": "legal",
      "reasons": [],
      "viableTargetPorts": [
        {
          "portId": 3712,
          "portName": "Sold as",
          "direction": "in"
        }
      ]
    }
  ],
}

```

```
{
  "resourceId": 12933,
  "name": "OCM3",
  "resourceTypeId": 187,
  "verdict": "legal",
  "reasons": [],
  "viableTargetPorts": [
    {
      "portId": 3711,
      "portName": "Sold as",
      "direction": "in"
    }
  ]
},
"truncated": true,
"total": 3928
}
```

```
curl -s -X POST -H "Authorization: Bearer $TOKEN" -H 'Content-Type: application/json' \
-d
'{"sourceResourceId":13811,"targetResourceId":7369,"interactionTypeId":135,"sourcePortId":1503,"targetPortId":1529}' \
"$BASE/v1/resource-interactions"
```

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

```
{
  "interactionId": 2147,
  "sourceResourceId": 13811,
  "targetResourceId": 7369,
  "interactionTypeId": 135,
  "contextId": null
}
```

A field holds one choice: to change it, send `replaceInteractionId` with the connection it replaces.

8. The bid and its three fields

The bid is another `find-or-create` (`type_key`: `"cab_bid"`), a `PATCH` for its properties (for example `tax_rate_bp`), and three fields filled exactly as in step 7: `customer` (to the customer), `level` and `terms`. Read it back with `GET /v1/resources/$BID`; its `edges` list the three — see the converted bid in step 15.

9. 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 $TOKEN" -H 'Content-Type: application/json' \
-d '{"childResourceId":24788}' "$BASE/v1/resources/$BID/members"
```

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

```
{
  "componentId": 253767,
  "parentResourceId": 24787,
  "childResourceId": 24788,
  "componentRole": null,
  "sourceSlotId": null,
  "sourceRuleId": null,
  "componentStatus": "active",
  "statusChangedAt": "2026-10-20T04:44:03.377Z",
  "sortOrder": 0,
  "properties": {},
  "createdAt": "2026-10-20T04:44:03.377Z"
}
```

`childResourceId` is a JSON **number**. A design set's ten material and hardware fields are filled like any field (step 7); a line inherits each one unless it chooses its own.

10. Lines, each with its product

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 — legal-targets with `q=<the product's identifier>&limit=5`, then the attachment. A line never carries the product identifier itself; it points at the product.

11. Read the tree back

`GET /v1/resources/$SET/members` lists what a container holds:

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

```
[
  {
    "childResourceId": 24786,
    "childResourceTypeId": 3425,
    "childName": "Capture line",
    "childTypeName": "Bid Line",
    "componentStatus": "active",
    "createdAt": "2026-10-19T05:24:35.450Z",
    "roles": [],
    "enrollmentCount": 1,
    "applicableActions": []
  },
  {
    "childResourceId": 24789,
    "childResourceTypeId": 3425,
    "childName": "Capture line two",
    "childTypeName": "Bid Line",
    "componentStatus": "active",
    "createdAt": "2026-10-19T05:24:35.507Z",
    "roles": [],
    "enrollmentCount": 1,
    "applicableActions": []
  }
]
```

Member rows carry no status and no properties. To know each line's status, read each line (`GET /v1/resources/:id`) — one request per child.

12. Preview what a status move needs

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/$SET/requirements?transitionId=823"
```

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

```

{
  "requirements": [
    {
      "requirementId": 2395,
      "name": "Contents compatible",
      "blocking": "hard",
      "satisfied": true,
      "missing": [],
      "leaves": [
        {
          "leaf": "subtree_compat",
          "label": "No design set settings conflicts",
          "caption": "nothing inside receives a setting yet",
          "satisfied": true,
          "attestable": false,
          "actionLabel": null,
          "actor": null,
          "actorRoles": [],
          "payloadFields": [],
          "blockedByActor": false,
          "satisfiedEver": false,
          "mayEmitUnvocab": false,
          "violations": [],
          "violationCount": 0,
          "subtreeCompatVacuous": true
        }
      ]
    },
    {
      "requirementId": 1528,
      "name": "R-SET-FREEZE – all lines frozen",
      "blocking": "hard",
      "satisfied": false,
      "missing": [
        "All lines frozen"
      ],
      "leaves": [
        {
          "leaf": "members",
          "label": "All lines frozen",
          "caption": "0/1 members at required status",
          "satisfied": false,
          "attestable": false,
          "actionLabel": null,
          "actor": null,
          "actorRoles": [],
          "payloadFields": [],
          "blockedByActor": false,
          "satisfiedEver": false,
          "mayEmitUnvocab": false,
          "memberTotal": 1,
          "memberSatisfied": 0
        }
      ]
    }
  ]
}

```

13. Freeze bottom-up

A set freezes only when every line in it is frozen. Ask too early and the answer is 422 with the same checklist:

```
{
  "error": "Transition blocked by unmet requirements",
  "requirements": [
    {
      "requirementId": 2395,
      "name": "Contents compatible",
      "blocking": "hard",
      "satisfied": true,
      "missing": [],
      "leaves": [
        {
          "leaf": "subtree_compat",
          "label": "No design set settings conflicts",
          "caption": "nothing inside receives a setting yet",
          "satisfied": true,
          "attestable": false,
          "actionLabel": null,
          "actor": null,
          "actorRoles": [],
          "payloadFields": [],
          "blockedByActor": false,
          "satisfiedEver": false,
          "mayEmitUnvocab": false,
          "violations": [],
          "violationCount": 0,
          "subtreeCompatVacuous": true
        }
      ]
    },
    {
      "requirementId": 1528,
      "name": "R-SET-FREEZE — all lines frozen",
      "blocking": "hard",
      "satisfied": false,
      "missing": [
        "All lines frozen"
      ],
      "leaves": [
        {
          "leaf": "members",
          "label": "All lines frozen",
          "caption": "0/1 members at required status",
          "satisfied": false,
          "attestable": false,
          "actionLabel": null,
          "actor": null,
          "actorRoles": [],
          "payloadFields": [],
          "blockedByActor": false,
          "satisfiedEver": false,
          "mayEmitUnvocab": false,
          "memberTotal": 1,
          "memberSatisfied": 0
        }
      ]
    }
  ]
}
```

So freeze each line first:

```
curl -s -X PATCH -H "Authorization: Bearer $TOKEN" -H 'Content-Type: application/json' \
-d '{"toStatusId":1319}' "$BASE/v1/resources/$LINE/status"
```

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

```
{
  "resourceId": 24789,
  "fromStatusId": 1318,
  "statusId": 1319
}
```

Then the set. Freezing records a snapshot of the line — see `resource.captured` in [events.md](#). A frozen line is read-only, and there is no way back to Draft.

14. Submit

Record the totals your application computed, then move the bid to Sent:

```
curl -s -X POST -H "Authorization: Bearer $TOKEN" -H 'Content-Type: application/json' \
-d '{"eventType":"bid.sent","payload":
{"subtotal_cents":7001,"discount_cents":0,"delivery_cents":0,"tax_cents":578,"install_cents":0,"total_cents":7579,"tax_rate_bp":825}}' \
"$BASE/v1/resources/$BID/attestations"
```

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

```
{
  "eventUlid": "01M5BSZAZ3SV3132XNPYTKV77X",
  "eventType": "bid.sent",
  "resourceId": 24787,
  "actorResourceId": 24784
}
```

Then `PATCH /v1/resources/$BID/status` to Sent. 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. A sent bid is read-only.

15. Accept, convert, find the order

The customer's acceptance is the `bid.accepted` attestation; the bid then moves to Accepted on its own. Convert is a status move followed by the `bid.converted` attestation; ReSync then creates the order, a moment later. Read the bid until its `derivatives` lists it:

From `get-derivatives-after-convert.json` — status 200:

```

{
  "resourceId": "24787",
  "resourceTypeId": "176",
  "ownerResourceId": "5",
  "name": "Harbor Homes – kitchen",
  "identifier": null,
  "properties": {
    "tax_rate_bp": 825
  },
  "statusId": "1315",
  "isVerified": false,
  "createdAt": "2026-10-20T04:44:01.410Z",
  "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": "Converted",
  "statusColorCode": null,
  "icon": "box",
  "lifecycleId": 108,
  "canvasId": null,
  "derivedFrom": [],
  "derivatives": [
    {
      "resourceId": 24790,
      "name": "Harbor Homes – kitchen",
      "typeName": "Order",
      "createdAt": "2026-10-20T04:44:05.245Z",
      "archived": false
    }
  ],
  "derivedKeys": [],
  "edges": [
    {
      "interactionId": 2572,
      "typeName": "bid_for",
      "direction": "out",
      "otherResourceId": 24786,
      "otherResourceName": "Harbor Homes",
      "thisResourcePortId": 6700
    },
    {
      "interactionId": 2573,
      "typeName": "priced_at",
      "direction": "out",
      "otherResourceId": 13808,
      "otherResourceName": "Contractor",
      "thisResourcePortId": 6701
    }
  ],
  {

```

```

    "interactionId": 2574,
    "typeName": "pays_on",
    "direction": "out",
    "otherResourceId": 14013,
    "otherResourceName": "Net 30",
    "thisResourcePortId": 6702
  }
],
"confirmedKeys": [],
"readOnly": {
  "statusName": "Converted"
}
}

```

Things to know

1. **Every `find-or-create` binds one identifier in your external system** — one per resource you create that way. 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.
3. **Budget about 8 requests per line**, on top of roughly 130 for the bid itself (customer, bid, fields, a design set, freezes, Submit, accept, convert). A 40-line bid is about 450 requests.
4. **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.
5. **Member rows carry no status.** Reading a tree's statuses costs one `GET` per child.
6. **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.
7. **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`.
8. **Identifiers are unique per external system across every organization**, not only yours. Prefix them with something that is yours.
9. **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.

Where to go next

- [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 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.

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 two scope sets

Set	Grants	Applies to
read_types	list and fetch	GET doors
operate_types	change	PATCH , enrol, exit, attest, status

Operating implies reading. A type in `operate_types` does not need repeating in `read_types` ; the readable set is the union of the two.

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`.
- **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`.

A third list, `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.

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.

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 kiosk stations use — refuses an integration key. (Kiosk station credentials are a separate plane, not covered by this pack.) 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 kiosk station 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/13813"
```

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.

4 Endpoints

Every door a key can open, and nothing else. Endpoints that exist but need a signed-in person, or a station credential, are not listed — they will return `401` or `403` for you no matter what you send.

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>find_or_create_types</code> .
<code>owner</code>	object null	<code>{ resourceId, name }</code> — the organization that owns the key.
<code>actsAt</code>	object null	Always <code>null</code> for a service key; used only by station credentials.

Errors: `401 invalid_api_key`.

From `self.json` — status 200:

```

{
  "apiKeyId": 781,
  "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:

Field	Type	Meaning
<code>nodes</code>	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.
<code>interactions</code>	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.
<code>content</code>	object	<code>{ resourceId, name, statusName, statusColorCode }</code> for the resource itself.
<code>alerts</code>	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.
<code>summary</code>	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 }</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 / source</code> shaped like a port <code>setting</code> 's). The icon to draw is the node's own <code>icon</code> .
<code>overlay</code>	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.
<code>tray</code>	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.
<code>view</code>	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: an unreadable type 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."
}
```

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 <code>403</code> .
<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: <code>400 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.**

Beyond the list-row fields (`icon` included), this returns `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), and `edges`. 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`).

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": "13813",
  "resourceTypeId": "1179",
  "ownerResourceId": "5",
  "name": "MSDEV Customer 2",
  "identifier": null,
  "properties": {
    "kind": "contractor"
  },
  "statusId": null,
  "isVerified": false,
  "createdAt": "2026-09-13T17:58:40.818Z",
  "archivedAt": null,
  "derivedFromTypeId": null,
  "resourceTypeName": "Customer",
  "propertySchema": {
    "kind": {
      "type": "select",
      "label": "Kind",
      "options": [
        "dealer",
        "high_volume_contractor",
        "contractor",
        "homeowner",
        "internal"
      ]
    }
  },
  "declaredPropertySchema": {
    "kind": {
      "type": "select",
      "label": "Kind",
      "options": [
        "dealer",
        "high_volume_contractor",
        "contractor",
        "homeowner",
        "internal"
      ]
    }
  },
  "orgName": "Popular Design Mill & Cabinet, Inc.",
  "ownerOrgId": "1",
  "canvasMode": null,
  "statusName": null,
  "statusColorCode": null,
  "lifecycleId": null,
  "canvasId": null,
  "derivedFrom": [],
  "derivatives": [],
  "derivedKeys": [],
  "edges": [
    {
      "interactionId": 104,
      "typeName": "priced_at",
      "direction": "out",
      "otherResourceId": 13808,
      "otherResourceName": "Contractor",
      "thisResourcePortId": 80
    },
    {
      "interactionId": 105,
      "typeName": "bid_for",
      "direction": "in",
      "otherResourceId": 13416,
      "otherResourceName": "MSDEV Bid 2",

```

```

    "thisResourcePortId": 79
  },
  ],
  "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": 24786,
    "childResourceTypeId": 3425,
    "childName": "Capture line",
    "childTypeName": "Bid Line",
    "componentStatus": "active",
    "createdAt": "2026-10-19T05:24:35.450Z",
    "roles": [],
    "enrollmentCount": 1,
    "applicableActions": []
  },
  {
    "childResourceId": 24789,
    "childResourceTypeId": 3425,
    "childName": "Capture line two",
    "childTypeName": "Bid Line",
    "componentStatus": "active",
    "createdAt": "2026-10-19T05:24:35.507Z",
    "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": "24787",
      "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": "24790",
    "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.

GET /v1/resources/:id/capabilities

The actions declared on this resource. **Needs read scope.** Response: { capabilities: [...] }.

GET /v1/resources/:id/requirements

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

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

Response: { requirements: [...] } — the same array a blocked status change returns, so one shape to learn. A full capture is requirements-read.json in the fixture set; the blocked status change is shown under PATCH /v1/resources/:id/status.

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	Kiosk station credentials only. An integration 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 actorAliases 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": 24787,
  "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": 253767,
  "parentResourceId": 24787,
  "childResourceId": 24788,
  "componentRole": null,
  "sourceSlotId": null,
  "sourceRuleId": null,
  "componentStatus": "active",
  "statusChangedAt": "2026-10-20T04:44:03.377Z",
  "sortOrder": 0,
  "properties": {},
  "createdAt": "2026-10-20T04:44:03.377Z"
}
```

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 PATCH.

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

From `exit.json` — status 200:

```
{
  "exited": true,
  "componentId": 253768
}
```

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
<code>fromContainerId</code>	number	Required. The container it is in now.
<code>toContainerId</code>	number	Required. The container it goes to. Must differ from <code>fromContainerId</code> .
<code>actorResourceId</code> / <code>actorAlias</code>		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": 253767,
  "enrolledComponentId": 253769
}
```

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 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 <code>403 subject_outside_grant</code> and nothing is recorded.
actorResourceId / actorAlias		Optional, as for <code>PATCH</code> .

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

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

```
{
  "eventId": "01M5BSZAZ3SV3132XNPYTKV77X",
  "eventType": "bid.sent",
  "resourceId": 24787,
  "actorResourceId": 24784
}
```

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": 24787,
  "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 operate scope** — a status move is an *attest* act, and your operate set grants it (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": 24789,
  "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": 2395,
      "name": "Contents compatible",
      "blocking": "hard",
      "satisfied": true,
      "missing": [],
      "leaves": [
        {
          "leaf": "subtree_compat",
          "label": "No design set settings conflicts",
          "caption": "nothing inside receives a setting yet",
          "satisfied": true,
          "attestable": false,
          "actionLabel": null,
          "actor": null,
          "actorRoles": [],
          "payloadFields": [],
          "blockedByActor": false,
          "satisfiedEver": false,
          "mayEmitUnvocab": false,
          "violations": [],
          "violationCount": 0,
          "subtreeCompatVacuous": true
        }
      ]
    },
    {
      "requirementId": 1528,
      "name": "R-SET-FREEZE – all lines frozen",
      "blocking": "hard",
      "satisfied": false,
      "missing": [
        "All lines frozen"
      ],
      "leaves": [
        {
          "leaf": "members",
          "label": "All lines frozen",
          "caption": "0/1 members at required status",
          "satisfied": false,
          "attestable": false,
          "actionLabel": null,
          "actor": null,
          "actorRoles": [],
          "payloadFields": [],
          "blockedByActor": false,
          "satisfiedEver": false,
          "mayEmitUnvocab": false,
          "memberTotal": 1,
          "memberSatisfied": 0
        }
      ]
    }
  ]
}

```

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"
}
```

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`. Like attestations, this door returns sentences in `error` rather than codes for its own refusals.

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": 7267, "name": "Laminate" },
  "scope": "inherited",
  "locked": true,
  "source": { "resourceId": 13811, "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. `legal-targets-q.json` in the fixture set shows `q` alone.

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

```

{
  "sourcePort": {
    "portId": 6707,
    "portName": "product",
    "direction": "out",
    "maxConnections": 1,
    "held": 0,
    "capRemaining": 1
  },
  "candidates": [
    {
      "resourceId": 24699,
      "name": "WD Laminate Modern Vertical 30",
      "resourceTypeId": 191,
      "verdict": "legal",
      "reasons": [],
      "viableTargetPorts": [
        {
          "portId": 6623,
          "portName": "Sold as",
          "direction": "in"
        }
      ]
    },
    {
      "resourceId": 24697,
      "name": "WD Laminate Modern Vertical",
      "resourceTypeId": 191,
      "verdict": "legal",
      "reasons": [],
      "viableTargetPorts": [
        {
          "portId": 6622,
          "portName": "Sold as",
          "direction": "in"
        }
      ]
    },
    {
      "resourceId": 13812,
      "name": "MSDEV Line 2 W30-1/2*12",
      "resourceTypeId": 182,
      "verdict": "legal",
      "reasons": [],
      "viableTargetPorts": [
        {
          "portId": 6621,
          "portName": "Sold as",
          "direction": "in"
        }
      ]
    },
    {
      "resourceId": 12934,
      "name": "WOCM3",
      "resourceTypeId": 187,
      "verdict": "legal",
      "reasons": [],
      "viableTargetPorts": [
        {
          "portId": 3712,
          "portName": "Sold as",
          "direction": "in"
        }
      ]
    }
  ],
}

```

```

{
  "resourceId": 12933,
  "name": "OCM3",
  "resourceTypeId": 187,
  "verdict": "legal",
  "reasons": [],
  "viableTargetPorts": [
    {
      "portId": 3711,
      "portName": "Sold as",
      "direction": "in"
    }
  ]
},
"truncated": true,
"total": 3928
}

```

`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 full capture is `legal-targets-type.json` in the fixture set.

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

Returns 201 { `interactionId`, `sourceResourceId`, `targetResourceId`, `interactionTypeId`, `contextId` }.

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

```
{
  "interactionId": 2147,
  "sourceResourceId": 13811,
  "targetResourceId": 7369,
  "interactionTypeId": 135,
  "contextId": null
}
```

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

```
{
  "error": "compat_deny",
  "message": "Incompatible with Laminate (Species).",
  "denyEdgeId": 520,
  "layer": "value",
  "a": 7350,
  "b": 7267,
  "via": 13811
}
```

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. 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": 24786,
  "created": true
}
```

Called again with the same identifier:

From `foc-find.json` — status 200:

```
{
  "resource_id": 24786,
  "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).

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.

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.

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, parentTypeId, 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.

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.

5 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`) carry detail specific to that code.

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`) 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.
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.
acto r_not_ _read able	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

Code	Status	Meaning	What to do
			attributed to your key.
live_key_not_supported	403	Live updates (GET /v1/live) are held by signed-in people and kiosk stations only; an integration key cannot open a stream.	Poll the resource's events instead (GET /v1/resources/:id/events) with a stored cursor.
owner_forced_to_grant	403	You passed an ownerId that is not yours.	Drop the parameter — the filter is applied for you.
outside_acts_at_scope	403	Your key is a station key (it acts at one place, 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.

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
<code>containment_capacity</code>	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.
<code>compat_type_no_t_connectable</code>	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>).
<code>capture_shape</code>	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
<code>no_viable_target_port</code>	—	The candidate passes every rule, but none of its ports can take this connection (direction, accepted type or its own capacity).
<code>status_read_only</code>	—	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
<code>find_or_create_refused</code>	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.
<code>alias_bind_refused</code>	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 redeem 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> .

Transport

Code	Status	Meaning	What to do
rate_limit	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 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.

6 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/14211/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/14211/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": "01M5BSZ80EZ70B79005YCA0GB8",
      "event_type": "resource.created",
      "occurred_at": "2026-10-20T04:44:01.422Z",
      "actor": {
        "resource_id": 24784,
        "name": "bid-app-docs"
      },
      "display": {
        "name": "Harbor Homes – kitchen",
        "resource_type_id": 176,
        "text": "* created – Bid"
      },
      "producer_id": "bid-app-docs"
    },
    {
      "event_ulid": "01M5BSZ80YB87EJT0X8JQ1HSGY",
      "event_type": "status.changed",
      "occurred_at": "2026-10-20T04:44:01.438Z",
      "actor": {
        "resource_id": 24784,
        "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": "01M5BSZ86FC7TJ5X6AX036YKZ4",
      "event_type": "property.changed",
      "occurred_at": "2026-10-20T04:44:01.615Z",
      "actor": {
        "resource_id": 24784,
        "name": "bid-app-docs"
      },
      "display": {
        "key": "tax_rate_bp",
        "from": null,
        "to": 825,
        "text": "🔗 tax_rate_bp: – → 825"
      },
      "producer_id": "bid-app-docs"
    }
  ],
  "nextCursor": "c1MTE2MzgyMy4yMjUyOQ"
}
```

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 (icon, keyProperties)</code> , never the values. In the

Event type	When
<code>ile_changed</code>	<code>all</code> and <code>changes</code> filters; never unread.
<code>resource_type.config_change_d</code>	A type's configuration changed. Today: an emission was declared or removed (<code>kind: "emission"</code> , <code>verb created / deleted</code>). Recorded on the type's resource only; carries <code>kind</code> , <code>verb</code> , <code>id</code> , <code>typeId</code> , <code>attestationEventTypeId</code> and <code>eventTypeKey</code> (the event type's code). In the <code>all</code> and <code>changes</code> filters; never unread.
<code>resource.captured</code>	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. <code>display</code> carries <code>transitionId</code> , <code>fromStatusId</code> , <code>toStatusId</code> , <code>properties</code> (the resource's own values the transition named) and, when something could not be recorded because the resource's owner cannot read it, <code>unreadable</code> (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 <code>display</code> , since they name other resources. In the <code>all</code> and <code>changes</code> filters; never unread.
<code>thread.created</code>	A conversation was started. Recorded on the conversation (<code>subject_key thread:{id}</code>) and, when it belongs to a resource, scoped to that resource — so it can appear in that resource's events. Carries <code>thread_id</code> , <code>creator_resource_id</code> , <code>context_resource_id</code> . In the <code>all</code> and <code>changes</code> filters; never unread.
<code>thread.participant_added / thread.participant_left</code>	A person joined or left a conversation (<code>self: true</code> when they left on their own). Same placement as <code>thread.created</code> ; carries <code>thread_id</code> and <code>participant_resource_id</code> .
<code>attestation.refused</code>	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 <code>eventType</code> , <code>code</code> (<code>attestation_status_gate_unmet</code> or <code>attestation_not_repeatable</code>) and <code>producerEventId</code> . 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.
<code>lifecycle.config_changed</code>	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 <code>kind</code> , <code>verb</code> , <code>id</code> , <code>changed</code> , <code>before / after</code> of the changed fields and the ids it touched (<code>statusId</code> , <code>transitionId</code> , <code>requirementIds</code> ...) — typed words such as names and labels are noted as changed, not copied. In the <code>quiet</code> filter only.
<code>consequence_rule.changed</code>	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 <code>verb</code> , <code>id</code> , <code>changed</code> , <code>before / after</code> and the rule's ids (<code>triggerEventType</code> , <code>subjectTypeId</code> , <code>mintTargetTypeId</code> ...). In the <code>quiet</code> filter only.
<code>derivation.refused</code>	A computed value or a templated name was not updated. Carries <code>key</code> and <code>code</code> — <code>unique_key_taken</code> , <code>status_read_only</code> , <code>derive_target_not_derivable</code> , or <code>setting_name_unreadable</code> (the name would use a setting value the resource's owner cannot read). In the <code>changes</code> filter.
<i>your attestation types</i>	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": "01M5BSZB3DC81MZPA01G0N139B",
      "event_type": "status.changed",
      "occurred_at": "2026-10-20T04:44:04.589Z",
      "actor": {
        "resource_id": 24784,
        "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": "01M5BSZB3B3T05393WCWM19QDJ",
      "event_type": "resource.captured",
      "occurred_at": "2026-10-20T04:44:04.587Z",
      "actor": {
        "resource_id": 24784,
        "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": "01M5BSZAZ3SV3132XNPYTKV77X",
      "event_type": "bid.sent",
      "occurred_at": "2026-10-20T04:44:04.451Z",
      "actor": {
        "resource_id": 24784,
        "name": "bid-app-docs"
      },
      "display": {
        "tax_cents": 578,
        "tax_rate_bp": 825,
        "total_cents": 7579,
        "install_cents": 0,
        "delivery_cents": 0,
        "discount_cents": 0,
        "subtotal_cents": 7001,
        "attested_action_label": "Send bid"
      },
      "producer_id": "bid-app-docs"
    }
  ],
  "nextCursor": "01M5BSZAZ3SV3132XNPYTKV77X",
  "afterCursor": "c1MTE2MzgZNy4yMjU00A"
}

```

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.

7 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.

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.

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.

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.

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.

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 kiosk stations receive live updates over a stream; that stream is not open to integration keys.

There is no bulk write. One resource per request.

8 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 · 2026-10-20 — the base URL and the docs site

Base URL. The API is served at `https://api.limnal.dev`. Every example in this pack now uses it; paths are unchanged (they already carry `/v1`). HTTPS only.

Docs site. The current version of this pack is published at `https://docs.limnal.dev`.

v1 · 2026-10-20 — a limit per key, births recorded under your key, a whole-bid quickstart

Rate limit per key. A key may now carry its own limit, set by ReSync (an administrator act — a key cannot change its own). Without one, the default stays 300 requests per minute. `x-ratelimit-limit` on every response, and the 429 message, say your key's own limit. Unknown and revoked keys always get the default.

Births carry your key's name. `producer_id` on `resource.created` and on a new resource's first `status.changed` is now your key's name when your request created the resource — by `find-or-create`, or a person first named through `actorAlias`. It used to read `api`. A resource ReSync creates in response to your write (the order a converted bid produces) carries `consequence.worker`. Rows written before today keep the value they were written with.

Quickstart rewritten as one whole bid, from an empty start to a converted order, with a real response for every door. [examples.http](#) follows the same walk. New captures in the pack: the type list and one type in full, a member enrolment (201), a requirements read, a status change (200 , 422 with the requirements, 409 for a move the lifecycle does not declare), an attestation (201 , 409 `attestation_not_repeatable` , 422

`attestation_status_gate_unmet`, `403 actor_not_readable`), a converted bid's `derivatives`, `400 ports_required`, `409 status_read_only`, `409 unique_key_taken`, a `429`, and an event history with `resource.captured`.

Corrections — the pack described these wrongly; the API did not change:

- An unknown query parameter is **ignored**, not refused (a known one a door cannot apply to a key, such as `sort`, is still refused). [conventions.md](#), [endpoints.md](#).
- The rate-limit window is **fixed** (it starts with your first request), not rolling; `retry-after` and `x-ratelimit-reset` are the seconds left in it.
- `GET /v1/resources/search` returns at most 20 rows **in all**, not per requested type.
- `GET /v1/aliases/resolve`'s `resourceType` is the type's display name, not its code.
- `GET /v1/resources/:id/events` also accepts `filter=quiet`.
- Per-door error lists now include `409 status_read_only` (`PATCH`, `members`, `exit`, `move`, `connections`), `409 unique_key_taken` (`PATCH`, `find-or-create`) and `403 actor_not_readable` (every door that takes an actor).
- `PATCH /v1/resources/:id/status` documents its `200` body and its `422` body, and that its operate scope is the *attest* act.
- Containers, cross-type search and move are open to keys (they have been since 2026-10-19); [conventions.md](#) still listed them as signed-in only.
- `external_system` names a system ReSync already knows; it is not registered per organization, and identifiers are unique per system across every organization.
- [events.md](#) lists `lifecycle.config_changed` and `consequence_rule.changed` (`quiet` filter), and says who may be named as an actor.

v1 · 2026-10-19 — key-lane completeness

New doors for keys:

- `POST /v1/resources/:id/move` — move a resource from one container to another in one step. Needs operate scope on the resource and both containers; both halves are recorded together or not at all.
- `GET /v1/resources/:id/containers` — the containers a resource is inside (read scope; unreadable containers left out).
- `GET /v1/resources/search` — name search across types. `enrollInto`, `moveOf` and `moveFrom` list exactly what the act would accept and need operate scope on the resource they name. A key never searches people (`person` is dropped) and `restrictToOrgId` does not apply to it.
- `POST /v1/identity-claims/redeem` — documented for the first time. It now needs `bind_systems` to include the account's system: `403 alias_bind_refused`, checked before the code.

`POST /v1/resources/find-or-create`: optional `name` and the actor fields. Creation is one step: the resource, your identifier and `resource.created` are written together, and `resource.created` now carries the properties it was created with and `alias: { system, code }`. A new resource is placed by its type's enrolment rules. Newly refused: the `resync` system and built-in system types (`400`), an identifier already bound to a resource of another type (`409`, no resource named), a key with no owning organization (`403 owner_grant_required`), a blank `name` (`400 invalid_resource_update`).

GET /v1/resources/:id/ports/:portId/legal-targets : `q` narrows the candidates; optional `limit` (1–100) returns a page with `truncated` and `total`. Without `limit` the response is unchanged.

GET /v1/resources : a key list honours `tagId`, and refuses `sort` with `400 sort_not_supported` — it used to be ignored without notice.

By-id reads follow 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 outside your key's own list.

POST /v1/resources/:id/attestations : `scopeResourceId` must be a resource your key can read; otherwise `403 subject_outside_grant` and nothing is recorded.

Documented, unchanged: the actor fields on members, exit and move; the codes `replace_mismatch`, `outside_acts_at_scope` and `invalid_resource_update`; the full list of doors that answer some refusals with a sentence ([errors.md](#)).

New codes: `invalid_limit`, `invalid_tag_id`, `sort_not_supported`.

Correction: the 2026-10-07 entry "attestation rules, ingest refusals, money fields" says freezing a line requires its `bid.line_frozen` record. That requirement no longer exists: freezing a line photographs it instead (`resource.captured`, see the 2026-10-16 and 2026-10-17 entries).

v1 · 2026-10-17 — bid declarations

Changed declaration (tenant vocabulary): `bid.sent` declares `subtotal_cents`, `discount_cents`, `delivery_cents`, `tax_cents`, `install_cents`, `total_cents` and `tax_rate_bp` — `customer_pct_bp` is no longer declared and no longer needed. A bid now carries its own price level and payment terms, and sending it records them (a `resource.captured` event). Read the current declaration from `GET /v1/attestation-event-types`. See [events.md](#).

v1 · 2026-10-16 — declared captures and containment limits

New event type (additive): `resource.captured` — a transition can declare what to photograph when it fires (the resource's own properties, its settings, and properties of the resources attached at named ports); every firing records one `resource.captured` on the resource, whichever way the status changed. In the `all` and `changes` filters; never unread. See [events.md](#).

New field (additive) on `GET /v1/lifecycles/:id/graph`: each transition carries `capture` for a reader who can author the lifecycle — never for a key.

New error code: `capture_shape` (lifecycle authoring in the web application; a key never meets it).

Containment limits (additive): a type can declare how many containers of a given type its resources may be inside. `GET /v1/resource-types/:id` carries `containmentCaps`. **New error** on `POST /v1/resources/:id/members`: `422 containment_cap` — the resource is already inside as many containers of that type as its type allows. See [errors.md](#).

New filter value (additive): `filter=quiet` on `GET /v1/resources/:id/events`. Configuration entries (`lifecycle.config_changed`, `consequence_rule.changed`, `canvas.view_changed`, `status.attention_changed`) now sit in it and are no longer in `all` or `changes`. They were never unread.

v1 · 2026-10-16 — type configuration on the record

New event types (additive), recorded on a type's resource only, in the `all` and `changes` filters and never unread: `resource_type.profile_changed` (the type's icon or key properties changed; `typeId`, `changedKeys`) and `resource_type.config_changed` (an emission was declared or removed; `kind`, `verb`, `id`, `typeId`, `attestationEventTypeId`, `eventTypeKey`). Neither carries a value or a name. See [events.md](#).

New fields (additive) on `GET /v1/lifecycles/:id/graph`: each status carries `legalTargets` and `illegalTargets` (a status id → the sentence saying why a transition to it is not allowed). Both are empty unless the reader can author the lifecycle — always, for a key.

Changed content: a resource made by a consequence rule now has an `interaction.created` event for its link back to the resource it was made from, in the same moment as the consequence itself. Earlier such links have none.

v1 · 2026-10-15 — a lifecycle as one graph

New endpoint (additive): `GET /v1/lifecycles/:id/graph` — a lifecycle in one read. It carries the statuses in display order (from the entry status outward; statuses no transition reaches come last, flagged `reachable: false`), each transition with its requirements (name, blocking, active flag, and a display label per condition) and gate counts, the types that run the lifecycle, and the consequence rules on those types. A rule lists `linkedTransitionIds` — the transitions an active requirement of which needs the very event that triggers the rule — with a display sentence. Every human-readable string in the body is ready to show as is. The reference data rules apply (any valid key may read it); `lifecycle.can.author` is always `false` for a key.

v1 · 2026-10-15 — nested containment in the view; containment loops refused

Changed content in `GET /v1/api-keys/self/view`: `nodes` now carries containment at every depth, not only the view subject's direct members. Each node's `parentId` names the container it is drawn in (the subject, or a member that itself has members). A resource held by more than one drawn container appears once, under the one nearest the subject. A member you cannot read is left out together with everything inside it. Every drawn container that holds at least one drawn member carries a `rollup` for its own members (`total`, `shown`, `types`, `notReadable`); the subject's `rollup` is unchanged (present only when members were left out). `tray.noInteractions` now counts members with no connections across every drawn container.

New error on `POST /v1/resources/:id/members`: `422 containment_cycle` — the resource being added is the container, or already contains it. See [errors.md](#).

v1 · 2026-10-14 — field attachments drawn as settings, not connections

Changed content in `GET /v1/api-keys/self/view`: a field attachment (a value set on a field port) is no longer listed in `interactions`, and a resource reachable only through one is no longer in `nodes`. The value is carried by the port's `setting` in `summary.nodes[id].ports`. The attachment itself is unchanged — `GET /v1/resources/:id/ports` and the connection doors still see it, and a member holding one still counts as `connected`.

New field (additive) in the same body's `summary.nodes[id]: inherited` — the field values a node inherits from an enclosing item, for kinds it has no field of its own. See [endpoints.md](#).

v1 · 2026-10-14 — field settings: deviation flag, words for what you cannot read

New response field (additive) in `GET /v1/resources/:id/ports` → `setting: deviates` — `true` when a field's own value differs from the inherited value it replaces, `false` otherwise (a difference from a lock above is a conflict, never a deviation). See [endpoints.md](#).

Changed content in the same `setting`: a value or source your key cannot read no longer comes back with `name: null`. Its `name` is the word a `setting` (a value) or an `enclosing item` (a source) and the object carries `readable: false`; `resourceId` stays `null`. Show `name` as-is.

v1 · 2026-10-14 — compatibility candidates and labels, boundary-port doors removed

New field `label` on each type of `GET /v1/compat/interaction-types`: the type's display label for a chooser (its forward label, first letter capitalized, else its name in plain words). Additive; session only.

New door (session only) `GET /v1/resources/:id/compat-candidates`: not available to keys — a key gets `403 compat_authoring_session_only`, the same refusal a key gets when it tries to author a compatibility marking.

Removed (session only, 0 callers): the seven canvas boundary-port doors under `/v1/canvases/:id/boundary-ports` (list, create, save positions, rename, delete, add a type, remove a type). None was ever available to keys.

v1 · 2026-10-14 — port type choices, view overlay, list order

New field `typeChoices` on each port of `GET /v1/resources/:id/ports`: the types that may still be added to the port's accepted types (readable, accepted by the port's definition, not already on it), each `{ resourceId, name, icon }`. **New optional parameter** `?workspace=` on the same door narrows those choices to global types and one owner's.

Changed content: `overlay.nodes` on `GET /v1/api-keys/self/view` now holds position rows only for drawn nodes; a row for a node the view hides (or a type filter leaves out) is no longer sent.

Changed order (no shape change): the nodes of `GET /v1/api-keys/self/view` are now in list order — the resource itself, then nodes whose status attention is `alert`, then `watch`, then the rest, each group in the graph's own order.

v1 · 2026-10-14 — type icons

New field `icon` on each row of `GET /v1/resources`, on `GET /v1/resources/:id`, and on each readable node of `GET /v1/api-keys/self/view` (also on the rollup's per-type rows there): the resource type's icon, always one of the fixed icon names listed under `GET /v1/resources`. It is the nearest icon the type or an ancestor type declares, counting only types your key can read, and `box` when there is none — so there is always something to draw.

Changed content: a type's presentation profile in the view's `summary` now reports `icon: "box"` instead of `null` when no readable type in its chain declares one.

v1 · 2026-10-13 — hidden nodes and the tray in the view

Changed content in `GET /v1/api-keys/self/view`: nodes the default view hides are left out of `nodes`, together with their connections. They still count in `alerts`, in `memberCount` and in the rollup.

New fields: a top-level `tray` (`{ noInteractions, hidden }`), and `connected` (`true · false`) on each drawn member and outside node.

v1 · 2026-10-13 — the view's logic layer setting

New field in a view's settings (`GET /v1/api-keys/self/view` → `view.settings`): `showLogic` (`true · false`; `false` = the view draws structure only). It is a setting for your renderer; this endpoint's body is the same either way.

v1 · 2026-10-13 — attention, the alert list, type profiles in the view

New fields in `GET /v1/api-keys/self/view`: each readable node's `statusAttention` (`none · watch · alert`) and `statusShape` (`circle · square · triangle · diamond`, or `null` = the shape for its attention); a top-level `alerts` array (the `watch` / `alert` statuses among the resource's readable members, with counts and up to five sample resource ids); in `summary`, a type's `profile` (`{ icon, keyProperties }`) and a node's `keys`.

v1 · 2026-10-12 — the resource a key acts at, through its default view

New endpoint: `GET /v1/api-keys/self/view` returns the resource your key acts at as its default view shows it — the resource's graph (readable members, connections, per-node summary, stored layout) with the view's settings applied, plus `view` ({ `name`, `settings` }, or `null` when the resource has no default view). Takes no parameters.

New code: `no_acting_resource` (404) — the key acts at no resource.

New field in a view's settings: `density` (`comfortable` · `compact` · `large` ; `null` = `comfortable`).

v1 · 2026-10-12 — a resource's `canvasId` prefers its default canvas

Changed content: in `GET /v1/resources/:id`, `canvasId` now resolves deterministically when a resource has more than one stored canvas: the resource's own forked canvas first, then the canvas marked as its **default**, then the oldest. Before, the default was not considered. A resource with no default canvas returns the same value as before. No field changes type, and `canvasId` stays `null` for a resource with no canvas.

v1 · 2026-10-12 — legal-targets predicts read-only refusals

Changed content: `GET /v1/resources/:id/ports/:portId/legal-targets` no longer lists a connection whose source sits in a read-only status. Such a candidate is `illegal` with the reason `status_read_only` — the code a create already returns (409) — and an empty `viableTargetPorts`. When the anchor is the read-only source, every candidate carries it. No field changes type. See [errors](#).

v1 · 2026-10-12 — connections must face the right way

New error code: `POST /v1/resource-interactions` refuses a connection between two ports that face the wrong way with 422 `port_direction`. The source port must send (`out` or `bidirectional`) and the target port must receive (`in` or `bidirectional`); `out` → `out`, `in` → `in` and `in` → `out` are refused. A connection with only one port, or none, is not affected. See [errors](#).

No change to `legal-targets` : it already listed only ports that face the right way. A create now enforces the same rule.

v1 · 2026-10-10 — names you cannot read

Behaviour change, no field changes type: a response no longer names a resource, person, organization or type your key cannot read. Where it would, the field keeps its type and reads a neutral word — `someone`, `a resource`, `a type`, or the positional words `an enclosing item`, `an item inside`, `a setting` — and the id beside it is kept. See [conventions](#).

New response fields (additive):

- `readable: false` on a row that names something you cannot read: `GET /v1/resources/:id` `edges[]`, `derivedFrom[]` and `derivedKeys[]`; an event's `actor` and the resources in its `display`; `sourceReadable` / `targetReadable` on a connection event's `display`.

Changed content:

- `GET /v1/resources/:id` — a `derivatives` row you cannot read is left out; a type you cannot read reports `resourceTypeName: "a type"` with an empty `propertySchema` and `declaredPropertySchema`.
- `GET /v1/resources/:id/events` — `producer_id` is another integration's key name only when you can read its service identity; for the event types the platform writes, `display` carries a fixed set of keys (unlisted stored keys are left out), and a connected app's key name and scopes appear only to administrators of its owner.
- Requirement labels (`/requirements`, `/advancement`, the 422 of a status change) name only what you can read.
- `GET /v1/resources/:id/events` — a connection made or removed between this resource and one you cannot read now appears on this resource's history (`interaction.created` / `interaction.deleted`), the other end as a resource with `sourceReadable` / `targetReadable: false`. It used to be left out.
- New event code: `derivation.refused` with `code: setting_name_unreadable`. See [events.md](#).

Documented: the `compat_deny` attribution fields on `legal-targets` reasons (`denyEdgeId`, `layer`, `firingNeighborId`, `firingNeighborName`, `firingNeighborTypeName`). They were already present. See [endpoints.md](#).

v1 · 2026-10-10 — field flag on ports, inherited property schema

New response fields (additive):

- `GET /v1/resources/:id/ports` — every port carries `isField: true` on a field (a port that points out, takes one connection and accepts exactly one type), `false` otherwise. It is the same rule that decides which ports carry a `setting`. See [endpoints.md](#).
 - `GET /v1/resources/:id` — `declaredPropertySchema`: the resource type's own `propertySchema` merged with every ancestor type's, a key declared nearer the resource's own type winning. `propertySchema` is unchanged.
-

v1 · 2026-10-10 — legal targets list only what you can read

Behaviour change: `GET /v1/resources/:id/ports/:portId/legal-targets` with `scope=context` (the default) now lists only the resources in `contextId` that you can read — the same rule `scope=type` already applied. A resource you cannot read was never a target you could connect to: creating the connection already refused it. Nothing else in the response changes. See [endpoints.md](#).

v1 · 2026-10-09 — conversation events

New event types (see [events.md](#)): `thread.created`, `thread.participant_added` and `thread.participant_left` — a conversation's lifecycle, now recorded. A conversation that belongs to a resource is scoped to it, so these can appear in that resource's events (`all` and `changes` filters). They never count as unread.

v1 · 2026-10-08 — polling for changes, new event types

Poll for changes (additive): `GET /v1/resources/:id/events` takes a new `after` parameter — events oldest first, from a cursor you stored, with `nextCursor` always present as the start of your next poll (`after=start` begins at the first event). A poll never skips an event, even one whose write started earlier and finished later. Every newest-first read now also returns `afterCursor`, the cursor to start polling from. A cursor this API did not issue answers the existing `400 invalid_cursor`. See [events.md](#).

New event types (see [events.md](#)): `resource.tagged` / `resource.untagged`, `resource.owner_changed`, `port.created` / `port.updated` / `port.deleted`, and `port_definition.changed`. The port types appear in the `changes` filter only.

New code: `403 live_key_not_supported` — the live-update stream (`GET /v1/live`) serves ReSync's web app and kiosk stations; an integration key is refused and polls instead. The stream's other codes (`live_unavailable`, `live_connection_not_found`, `live_connection_limit`, `live_view_limit`, `invalid_views`) cannot be reached with an integration key; they are noted here for completeness.

v1 · 2026-10-07 — attestation rules, ingest refusals, money fields

Repeatable is now enforced (see [endpoints.md](#)): an attestation type that is not declared `repeatable` is recorded once per resource per status. A second one in the same status answers the new `409 attestation_not_repeatable` (see [errors.md](#)); it can be recorded again after the resource changes status. Types whose acts genuinely recur are declared repeatable.

Status rules on the bid attestations: `bid.sent` can be recorded only on a bid in Draft, `bid.accepted` only on a Sent bid, and `bid.line_frozen` only on a line in Draft; `422 attestation_status_gate_unmet` otherwise. Freezing a line now also requires its `bid.line_frozen` record (a requirement on the move).

Event ingest: a declared attestation type sent through ingest now obeys the same status and repeat rules. A refused event is recorded as `attestation.refused` in its place (see [events.md](#)); a resent duplicate is still just a duplicate. The `202` response is unchanged.

New property attribute (additive): a type's `propertySchema` field may carry `derivable: false` — that property (typically money) is never computed by a formula; you enter it.

Requirement labels: an evidence condition without its own label is now named by its declared action ("Send bid") instead of a reworded event type ("Bid sent").

New response field (additive): `GET /v1/resources/:id` carries `readOnly` — `null`, or `{ statusName }` when the resource is in a read-only status. Use it to show the resource as read-only instead of letting a write be refused.

v1 · 2026-10-07 — read-only statuses

New error code (see [errors.md](#)):

- `status_read_only` (409) — a lifecycle can declare a status **read-only**: a resource sitting in it keeps its own state. Writes to its properties, name or identifier, enrolling or removing members it contains, creating, retyping or deleting a connection it is the source of, and un-confirming a property are refused. The body carries `resourceId` and `statusId`; `message` is a sentence with no ids. Advancing its status, attestations, archiving, connections that point *at* it, and confirming a property are unaffected.

New response fields (additive):

- `GET /v1/statuses` — each status carries `subjectReadOnly`.
- The available moves (`GET /v1/state-transitions`, and `transitions[]` on `/advancement`) carry `toStatusSubjectReadOnly`: whether taking that move makes the resource read-only.
- `GET /v1/attestation-event-types` — each type carries `requiredSubjectStatusId`: the status a resource must be in for that event to be attested (`null` = any). Attesting it in another status answers `422 attestation_status_gate_unmet`, as before; the check now happens atomically with the append.

v1 · 2026-10-07 — readable requirement conditions

New response field (additive; see [endpoints.md](#)):

- Every entry in a requirement's `leaves[]` now carries `caption` — where that condition stands for this resource, in words (`"missing"` / `"present"`, `"not recorded yet"` / `"recorded"`, `"2/3 members at required status"`, `"everything inside fits"`). It appears wherever `leaves[]` does: `/requirements`, `/advancement`, and the `422` body of a blocked `PATCH /v1/resources/:id/status`.
- A leaf's `label`, and the requirement's `missing[]`, now name things instead of numbering them — for example `"Bid for connection to a Customer"` rather than an interaction-type number. Show both as given.

Message change: the `message` of `unique_key_taken` now reads, for example, `"W3012 is already used by another catalog product owned by Popular Design Mill & Cabinet, Inc."` and no longer carries a resource number. Use `holderResourceId` to reach the holder.

v1 · 2026-10-06 — declared unique keys

New error code (additive; see [errors.md](#)):

- `unique_key_taken` (409) — a resource's type, or one of its parent types, can declare a **unique key**: its `identifier`, or one property. Within one owner, no two live resources of that type family may hold the same value. For example, a product SKU is unique across every product category, and a modification's `sku` is unique among modifications. Any write that would create a second holder is refused with this code: a create (`POST /v1/resources/find-or-create`), a property or identifier change (`PATCH /v1/resources/:id`), or an unarchive. `message` names the holder. `holderResourceId`, `key` and `value` say which. An archived resource releases its key. Empty values are never compared.

A find-or-create that is refused this way leaves nothing behind: no resource and no alias.

v1 · 2026-10-05 — settings and allowed interaction types on ports

New response fields (additive; see [endpoints.md](#)):

- `GET /v1/resources/:id/ports` — every port gains `allowedInteractionTypes` (`[{ id, name }]`, the types a connection on that port may use) and `setting` (on a field: the value it resolves to, where it comes from, what it overrides or conflicts with, a display `note`, and — on a field holding its own value — `clear` (`{ label, edgeId }`: delete that connection to fall back to the inherited value); `null` elsewhere). Resources your key cannot read are never named in a `setting`.
- `GET /v1/interaction-types` — every type gains `connectable` (`false` for the compatibility types).

New code: `compat_type_not_connectable` (422) — a compatibility type used for a connection on a port.

Behaviour, no shape change:

- `PATCH /v1/resources/:id/status`: when another request moves the resource to the same status at the same moment, the second returns `200` and records nothing; the change is recorded once.
- `DELETE /v1/resource-interactions/:id`: deleting a connection that is already gone returns the existing `404 interaction_not_found`; only the deletion that happened is recorded.
- `POST /v1/resource-interactions`: of two simultaneous connections to a port that takes one, the second gets the existing `422 port_max_connections`. A `replaceInteractionId` already replaced by another request gets the existing `400 replace_mismatch`.

v1 · 2026-10-04 — ports and attachments

Four new doors (see [endpoints.md](#)):

- `GET /v1/resources/:id/ports` — a resource's ports, including `contentsScope`.
- `GET /v1/resources/:id/ports/:portId/legal-targets` — what a port could connect to, and why not.
- `POST /v1/resource-interactions` — make an attachment. Needs operate scope on the choosing side and read scope on the chosen side.
- `DELETE /v1/resource-interactions/:id` — remove one.

New codes, all in [errors.md](#): `port_max_connections`, `port_inactive`, `accepted_type`, `compat_authoring_session_only`, `port_not_found`, `port_not_on_resource`, `ports_required`, `self_interaction`, `source_resource_id_required`, `target_resource_id_required`, `interaction_type_id_required`, `invalid_interaction_id`, `interaction_not_found`, `context_id_required`, `invalid_context_id`, `exclude_mismatch`, and the advisory `no_viable_target_port`. `compat_deny` and `contents_locked` now also come from `POST /v1/resource-interactions`. Compatibility markings stay with signed-in people.

v1 · 2026-10-03 — a requirement that checks a resource's contents

No existing field changes. Requirements may now contain a condition with `leaf: "subtree_compat"`: everything inside the resource must fit the settings passed down to it, and nothing may break a locked setting. Wherever requirement leaves appear — `GET /v1/resources/:id/requirements`, `GET /v1/resources/:id/advancement`, and the 422 body of `PATCH /v1/resources/:id/status` — such a leaf carries three new fields: `violationCount`, `violations` (up to 10) and `subtreeCompatVacuous`. See [endpoints.md](#). Clients that branch on `leaf` should treat an unknown value as a non-attestable, server-evaluated condition.

v1 · 2026-10-03 — settings that pass down to contents

No request or response field changes. A container may now pass a setting down to the resources inside it. `POST /v1/resources/:id/members` checks the resource you add — and everything inside it — against the settings it would receive, and refuses a conflict with one of two new codes (see [errors.md](#)):

- 422 `compat_deny` — the resource, or something inside it, is marked incompatible with a setting it would receive.
- 422 `contents_locked` — the container locks a setting and the resource, or something inside it, uses a different value.

Both may carry `viaAncestorId`, `offendingResourceId` and `offenderCount`; an id is omitted when your key cannot read that resource. Nothing is written when either is returned.

v1 · 2026-10-03 — more precise refusals on four key paths

No request or response field changes. Four paths now refuse, or attribute, more precisely:

- `GET /v1/state-transitions` with `resourceId` requires that your key can read the resource. A resource in another organization returns 403 `subject_outside_grant`, and a missing id returns 404 `resource_not_found`, as for `GET /v1/resources/:id`. The `lifecycleId` form is unchanged.
- `actorResourceId` on `PATCH /v1/resources/:id`, `POST /v1/resources/:id/attestations` and `PATCH /v1/resources/:id/status` must name a person your key can read. Otherwise the request is refused with the new code 403 `actor_not_readable` (see [errors.md](#)). `actorAlias` and the default (your key's own identity) are unchanged.
- `POST /v1/resources/find-or-create`: a resource it creates now records your key's service identity as the actor of its birth event, instead of no actor.

- Event ingest: an event whose subject identifier resolves to a resource your key may not attest on is accepted (202) but is not attached to that resource. This was already true for `resource:{id}` subjects and now holds for every identifier. A person first seen through your key is created under your organization.

v1 · 2026-09-29 — authorization by grant

No request or response shape changes. Access is now decided from explicit grants on both sides of the API; for a key, those grants are exactly what its scopes always described. What you can see is what you could see before, except the cases below, which the documentation never promised:

- `GET /v1/aliases/resolve` answers only for resources your key can read. An identifier bound to a resource in another organization now returns `404 alias_not_found`.
- `POST /v1/aliases/bind` requires the target to be a resource your key may change; otherwise `403 subject_outside_grant`. A `409` no longer names the resource the identifier is already bound to.
- `POST /v1/resources/find-or-create` returns an existing resource only if your key can read it. An identifier already bound to a resource you cannot read returns `409 find_or_create_refused`, without naming that resource.
- A **revoked** key sent to an endpoint that also accepts a signed-in person now returns `401 invalid_api_key` instead of the session layer's phrase.
- `POST /v1/resources/:id/attestations` and `PATCH /v1/resources/:id/status` check your access to the resource **before** validating the body, so a refused key gets the scope refusal, never a list of the resource's event types.
- `GET /v1/resources/:id` and `/advancement`: a missing id is `404 resource_not_found` for everyone.
- `verb_not_granted` and `key_unbound` are now emitted (see [errors.md](#)); a key never holds more than the person who provisioned it.
- The event-ingest endpoint authenticates with an API key; a producer's identity is its key.
- Members projection: on `GET /v1/orgs/:orgId/members`, each row's `membershipRole` is replaced by `access ( { grantSet, administer, verbs } )`, read from what the member has actually been granted. This endpoint is for signed-in people only and cannot be reached with an API key; it is noted here for completeness.

v1 · 2026-09-29 — no door changes; two refusal codes reserved (not yet emitted)

No endpoint, parameter or response shape changes. Two codes are added to [errors.md](#) ahead of use, so clients can classify them before they are ever returned:

- `verb_not_granted` (403) — the resource is within reach, the action is not.
- `key_unbound` (403) — the key has not been bound to an acting identity.

Neither is emitted yet. When they start to be, this changelog will say so.

v1 · 2026-09-25 — initial contract

First published contract for external developers.

Discovery

- `GET /v1/api-keys/self`

Resources — reading

- `GET /v1/resources` — list by type, with `include=properties`, keyset paging
- `GET /v1/resources/:id`
- `GET /v1/resources/:id/events`
- `GET /v1/resources/:id/members`
- `GET /v1/resources/:id/advancement`
- `GET /v1/resources/:id/capabilities`
- `GET /v1/resources/:id/requirements`

Resources — writing

- `PATCH /v1/resources/:id`
- `POST /v1/resources/:id/members`
- `POST /v1/resources/:id/exit`
- `POST /v1/resources/:id/attestations`
- `PATCH /v1/resources/:id/status`

Creating and addressing

- `POST /v1/resources/find-or-create`
- `POST /v1/aliases/bind`
- `GET /v1/aliases/resolve`

Reference data

- `GET /v1/resource-types`, `GET /v1/resource-types/:id`
- `GET /v1/statuses`
- `GET /v1/lifecycles`, `GET /v1/lifecycles/:id`
- `GET /v1/state-transitions`
- `GET /v1/attestation-event-types`
- `GET /v1/property-keys`
- `GET /v1/units`
- `GET /v1/interaction-types`

Across all of them

- Bearer authentication, scoped by owning organization plus a read set and an operate set
- 300 requests per minute per key, with `x-ratelimit-*` and `retry-after`
- `x-request-id` on every response, echoed if you send one
- Error envelope `{ error, message?, ...extras }` with stable codes