Shelfbot API Specification

SB-SPEC-API · Version 3.3 · 10 September 2026

1. Introduction

This document defines the wire protocol between a Warehouse Management System (WMS) and a Shelfbot Automated Storage & Retrieval System (ASRS) installation. It is intended for WMS integrators, systems architects, and integration test engineers responsible for connecting a customer WMS to a Shelfbot deployment.

The API is a bidirectional HTTP webhook protocol. The WMS owns business logic and customer-facing state (catalogue, orders, fulfilment records, authoritative stock counts). Shelfbot owns physical tote state, robot operation, scan validation, and the human-machine interface (HMI) at each station. The two systems exchange authoritative events to keep their views consistent.

This specification forms part of the Shelfbot integration contract. A WMS is compatible when it implements the WMS to Shelfbot endpoints correctly and acknowledges the event stream; which event types it processes is its own choice. Two integration profiles are documented in the Shelfbot IT Integration document: Shelfbot Simple (handle `order.completed` and use the read endpoints) and Shelfbot Full (process the complete event stream).

2. Purpose & Scope

2.1 Purpose

The purpose of this specification is to define the wire-level contract — transport, authentication, payload schemas, and event semantics — that a WMS must implement to integrate with a Shelfbot deployment. It aims to give integrators everything they need to build, test, and certify a compliant integration without needing to read Shelfbot source code.

2.2 Scope

This document covers:

- HTTP transport requirements and TLS policy
- Authentication, signing, and replay protection
- Idempotency and deduplication
- Batching rules and partial-failure semantics
- Response envelope and error-code vocabulary
- Retry policy and failure classification
- Endpoint definitions for WMS → Shelfbot (SKU sync, order lifecycle, carton feed, batch status)
- Endpoint definitions for reading current state (SKU list and summary, SKU detail, order state, bin contents, shipper lookup, batch lookup)
- Endpoint definitions for Shelfbot → WMS (order progress and shipper events, inventory events, exceptions)
- Batch and serial tracking semantics

This document does not cover:

- Shelfbot robot mechanical, electrical, or firmware specifications
- Shelfbot HMI user experience or operator workflows
- Shelfbot-internal concepts not exposed in the contract (totes, compartments, aisles, stations, scanner hardware)
- Warehouse racking, floor slab, fire engineering, or sprinkler design
- Customer-facing WMS UX
- Internal Shelfbot database schemas or module layout

For these topics, refer to the Shelfbot Racking Specification, the Shelfbot Electrical Specification, and the Shelfbot User Manual.

3. API Overview

Integration is a pair of HTTP webhook endpoints facing each other — the WMS hosts one set, Shelfbot hosts the other. All payloads are JSON and all requests are signed. Write operations use POST; Shelfbot also hosts a set of GET endpoints for reading current state (see *Reading current state*).

Role split

- **WMS is the source of truth** for: the SKU catalogue (what exists, what it's called, what it looks like, what hazards it carries); orders (what customers have requested, current fulfilment state); authoritative inventory counts at the SKU / batch / serial level; accounting / commercial workflows.
- **Shelfbot is the source of truth** for: physical tote state, robot and station availability, pick-in-progress state, scan validity. Bins are identified by `bin_id` in event payloads and MAY be read via the endpoints in *Reading current state*; compartment structure within a bin remains internal to Shelfbot and is never exposed in WMS-facing payloads. Shelfbot also decides which shipper each order is picked into and when a shipper is full; the WMS is told through shipper events and reads and never assigns shippers (see *Shippers*).

Neither side trusts the other's view of its own domain. On receipt of an event, each side validates against its own state and refuses invariant violations.

Scanning

All scanning is performed by Shelfbot-attached scanners at Shelfbot stations. Shelfbot owns scan validation end-to-end and is the source of all pick, stock-in, and adjustment events emitted to the WMS. The WMS does not operate scanners against Shelfbot stock and does not originate physical-state-changing events.

Shippers

A shipper is the container an order is picked into on the put-to-light trolley: a shipping carton, or a reusable order tote. To the API a shipper is nothing more than the label code the customer printed and stuck on it, bound to an order for a while. Shelfbot decides which order goes into which shipper as the runner inducts it, and when a shipper is full; an order may therefore be picked into one shipper or into several, one after another. The WMS is told what happened through the `shipper.opened` and `shipper.closed` events (*POST /v1/order/event*), the shipper list on the order read (*GET /v1/orders/*

{id}) and a lookup by label (*GET /v1/shippers/{shipper_id}*). Nothing about shippers is sent to Shelfbot: the WMS never assigns, sizes or counts them, and a shipper's kind and size are the customer's own knowledge, carried by its label, not part of this contract.

Labels may be reused. A label stays bound to its order until the WMS closes that order with `state: "fulfilled"` or `"cancelled"` (*POST /v1/order/update*); until then the lookup resolves the label to the order it holds, whether or not the shipper is still on the trolley. The API identity of a shipper is therefore the order, the label and `shipper_seq`, a counter Shelfbot assigns within the order (1 for its first shipper, 2 for the next), which keeps two uses of one label against the same order apart.

Directionality convention

An endpoint is named from the receiver's perspective. `/v1/sku/update` is hosted by Shelfbot and called by the WMS; `/v1/order/event` is hosted by the WMS and called by Shelfbot. *Endpoints — WMS → Shelfbot* defines the Shelfbot-hosted endpoints; *Endpoints — Shelfbot → WMS* defines the WMS-hosted endpoints. *Reading current state* defines additional Shelfbot-hosted GET endpoints, also called by the WMS.

4. Protocol

This section defines the protocol-level contract (transport, auth, idempotency, batching, envelope, and retry) that applies to every endpoint in this specification. One asymmetry is called out where it applies: requests into Shelfbot return the full response envelope with per-item results, while event deliveries to the WMS are acknowledged by any HTTP 2xx (see *Acknowledgement* under the Shelfbot to WMS endpoints).

4.1 Transport

Requirement	Value
Scheme	`https://` only — plain HTTP MUST be rejected
TLS	1.2 minimum, 1.3 preferred
Cert verification	Required. Per-deployment `insecure_skip_verify` is permitted for lab/dev only and MUST default to off
Method	`POST` for all write endpoints; the read endpoints in *Reading current state* use `GET`
Content-Type	`application/json; charset=utf-8` — receivers MUST return `415` otherwise
Max body size	1 MB — receivers MUST return `413 { error.code: "BODY_TOO_LARGE" }` above that
Request timeout	30s — sender aborts and treats as network failure (→ retry)
URL prefix	`/v1/` for all endpoints in this version

4.2 Authentication

Every request is signed with HMAC-SHA256 over the raw request body, plus a timestamp to prevent replay. For body-less GET requests the raw body is the empty string (see *Authentication and envelope* under *Reading current state*).

4.2.1 Required headers

Header	Example	Notes
`X-Shelfbot-Key-Id`	`wms-prod-1`	Identifier the receiver uses to look up the shared secret. Also implicitly identifies the sender.
`X-Shelfbot-Timestamp`	`1745366400`	Unix seconds (UTC) at the moment of signing.
`X-Shelfbot-Signature`	`sha256=4f2a…`	Lowercase hex HMAC-SHA256 of `<timestamp>.<raw_body>` using the shared secret.
`X-Shelfbot-Message-Id`	`01HZ…` (ULID/UUID)	Unique per request. Used for dedupe (see *Idempotency*).

4.2.2 Verification (receiver)

1. Reject if any required header is missing → `400 { error.code: "AUTH_MISSING_HEADER" }`.
2. Look up the secret for `X-Shelfbot-Key-Id`. Unknown key → `401 { error.code: "AUTH_UNKNOWN_KEY" }`.
3. Reject if `|now - X-Shelfbot-Timestamp| > 300s` → `401 { error.code: "AUTH_TIMESTAMP_SKEW" }`.
4. Recompute HMAC over `<timestamp>.<raw_body>` and compare in constant time. Mismatch → `401 { error.code: "AUTH_SIGNATURE_INVALID" }`.
5. IP allowlisting MAY be enforced at the network edge, reverse proxy, API gateway, firewall, or application layer. The implementation is deployment-specific and does not form part of the API contract; HMAC signatures over TLS are the authentication mechanism of the API. Allowlisting is available where required by customer security policy.

4.2.3 Identity

The sender is identified implicitly from `X-Shelfbot-Key-Id`. Payloads MUST NOT carry a source / vendor field — receivers derive sender identity from auth and use it for tagging (e.g. Shelfbot tags inventory rows with the WMS name mapped from the key id).

4.2.4 Secrets handling

- HMAC secrets MUST be stored in environment variables or a secrets manager. Never in version control.
- Secrets MUST be ≥32 random bytes.
- URLs MAY live in env vars or config files.

Recommended Shelfbot-side env var names:

```
WMS_BASE_URL          # e.g. https://wms.example.com
WMS_HMAC_KEY_ID       # identifier the WMS uses to look up its secret
WMS_HMAC_SECRET       # shared secret
```

Recommended WMS-side env var names (symmetric):

```
SHELFBOT_BASE_URL
SHELFBOT_HMAC_KEY_ID
SHELFBOT_HMAC_SECRET
```

4.2.5 Webhook URL discovery

Static config on both sides. Each peer's base URL + key id + secret is configured at deploy time. No runtime registration endpoint in v1.

4.3 Idempotency

- Every request carries `X-Shelfbot-Message-Id` (ULID or UUID recommended).
- Receivers MUST cache the result of each successfully-processed message id for 24 hours.
- A replay (same id within window) MUST return the cached response with the same status code and body, without re-processing.
- For batched requests (see *Batching*), the message id covers the entire batch. On Shelfbot-hosted endpoints a replay returns the cached per-item results array; for event deliveries to the WMS, a replay is acknowledged with 2xx like the original.

4.4 Batching

Every endpoint accepts an array, even for a single item. There is no single-item shorthand.

```
{ "skus": [ { ... }, { ... } ] }
```

Rule	Value
Max items per batch	100 — exceed → `400 { error.code: "BATCH_TOO_LARGE" }`
Dedupe scope	The whole batch shares one `X-Shelfbot-Message-Id`
Partial failure	Allowed. Returns `200 OK` ; per-item outcomes in `data.results[]`

4.4.1 Per-item result shape

```
{
  "ok": true,
  "message_id": "01HZ...",
  "data": {
    "results": [
      { "index": 0, "id": 1234567890, "ok": true },
      { "index": 1, "id": 1234567891, "ok": false,
        "error": { "code": "INVENTORY_NONZERO", "message": "...", "details": { "sku":
"BOLT_M5", "inventory": 4 } } }
    ]
  }
}
```

Sender retries only items where `ok=false` AND `error.code` is in the retryable set (see *Retry decision*).

Partial failure and per-item results apply to requests into Shelfbot. Event deliveries to the WMS are acknowledged as a whole batch with any HTTP 2xx response: see *Acknowledgement* under the Shelfbot to WMS endpoints.

4.5 Response envelope

Every response — success or error — uses the same envelope.

This envelope applies to responses from Shelfbot-hosted endpoints, both writes and reads. For event deliveries to the WMS, no response body is required and any body is ignored: see *Acknowledgement* under the Shelfbot to WMS endpoints.

4.5.1 Success

```
{
  "ok": true,
  "message_id": "01HZ...",
  "data": { ... }
}
```

4.5.2 Error

```
{
  "ok": false,
  "message_id": "01HZ...",
  "error": {
    "code": "INVENTORY_NONZERO",
    "message": "Cannot delete SKU with inventory > 0",
    "details": { "sku": "BOLT_M5", "inventory": 4 }
  }
}
```

4.5.3 Standard error codes

Code	HTTP	Meaning
`AUTH_MISSING_HEADER`	400	Required auth header absent
`AUTH_UNKNOWN_KEY`	401	`X-Shelfbot-Key-Id` not recognised
`AUTH_TIMESTAMP_SKEW`	401	Timestamp too far from receiver clock (>300s)
`AUTH_SIGNATURE_INVALID`	401	HMAC mismatch
`BODY_TOO_LARGE`	413	Request body > 1 MB
`BATCH_TOO_LARGE`	400	More than 100 items in batch
`BAD_REQUEST`	400	Malformed JSON or schema violation
`UNSUPPORTED_MEDIA_TYPE`	415	Content-Type not `application/json`
`INTERNAL_ERROR`	500	Receiver-side failure (DB down, etc.)

Per-item error codes are listed under each endpoint.

4.6 Retry policy (sender)

4.6.1 Cadence

Exponential backoff, 7 attempts: 30s, 1m, 5m, 15m, 1h, 6h, 24h (~32h total). After the final attempt, drop into a dead-letter queue for human review.

4.6.2 Retry decision

Outcome	Action
2xx	Success. Drop from queue. (For batches: retry only items with retryable per-item errors in a follow-up batch.)
408, 429, 5xx, network/TLS error	Retry per cadence above
4xx (other than 408/429)	Permanent failure. Don't retry. Log/alert.

429 responses SHOULD include a `Retry-After` header; the sender uses `max(Retry-After, next backoff)`.

5. Endpoints — WMS → Shelfbot

Shelfbot's data model is flat: every stockable item is a SKU. There is no product/variant hierarchy — each SKU stands alone with its own barcode, image, tags, and inventory.

The WMS-to-Shelfbot direction carries SKU catalogue sync and order lifecycle only. Physical-state changes (picks, stock-in, adjustments) always originate at Shelfbot and flow the other way (*Endpoints — Shelfbot → WMS*).

5.1 POST /v1/sku/update

Direction: WMS → Shelfbot **Purpose:** Create or update SKUs in Shelfbot. **Trigger:** WMS emits whenever a SKU is created or modified. Typical use is a 60s background sync.

Request:

```
{
  "skus": [
    {
      "sku": "BOLT_M5_10HST_BLACK",
      "title": "BOLT M5 10HST Black",
      "barcode": "5012345678900",
      "alt_barcodes": ["05012345678900", "SUP-BM5-10"],
      "supplier_sku": "SUP-BM5-10",
      "inventory_quantity": 0,
      "external_id": 1234567891,
      "tags": ["FASTENER", "METRIC"],
      "pick_strategy": "fifo",
      "hazard_classes": ["flammable"],
      "batch_tracked": false,
      "serial_tracked": false,
      "requires_expiry": false,
      "batch_scan": "per_receipt",
      "image": { "id": 1234567892, "src": "https://wms.example.com/img/bolt-m5.jpg" },
      "tote_division": 24,
      "each":  { "length_mm": 10, "width_mm": 5,  "height_mm": 5,  "weight_g": 2 },
```

```
      "inner": { "length_mm": 100, "width_mm": 50, "height_mm": 50, "weight_g": 25,
"quantity": 10 },
      "outer": { "length_mm": 300, "width_mm": 200,"height_mm": 200,"weight_g": 280,
"quantity": 100 }
    }
  ]
}
```

Fields:

Field	Type	Req	Meaning
`skus[].sku`	string	yes	Primary key. Case-sensitive identifier used everywhere Shelfbot references this item.
`skus[].title`	string	yes	Human-readable display name shown in the Shelfbot UI.
`skus[].barcode`	string	no	Primary scanned barcode value. Used to match a physical scan to this SKU. Empty string = no barcode.
`skus[].alt_barcodes`	string[]	no	Additional barcodes that also identify this SKU. Must not collide with another SKU's primary or alt barcodes.
`skus[].supplier_sku`	string	no	The supplier's own SKU code for this item. Opaque to Shelfbot.
`skus[].inventory_quantity`	int	no	WMS-side stock count, for reference only. Shelfbot tracks its own counts independently.
`skus[].external_id`	int|string	no	The WMS's own internal id for this SKU. Opaque to Shelfbot — stored and echoed back in events.
`skus[].tags`	string[]	no	Free-form classification tags. Shelfbot auto-adds the sender's identity tag (derived from the HMAC key id) on top of these.
`skus[].pick_strategy`	enum	no	One of `"fifo"`, `"lifo"`, `"fefo"`, `"random"`. `fefo` picks earliest expiry first, ties broken by receipt order, with undated stock sorted after every dated batch. Unset: `fefo` for SKUs with `requires_expiry: true`, otherwise the per-deployment default (typically fifo).
`skus[].hazard_classes`	string[]	no	Handling-rule classifications. Recommended vocabulary: `"flammable"`, `"corrosive"`, `"oxidizer"`, `"toxic"`, `"magnetic"`, `"fragile"`, `"esd"`, `"sharp"`, `"heavy"`.
`skus[].batch_tracked`	bool	no	Default false. If true, every stocked unit is associated with a batch number.
`skus[].serial_tracked`	bool	no	Default false. If true, every unit has a unique serial number that must be scanned individually.

Field	Type	Req	Meaning
`skus[].requires_expiry`	bool	no	Default false. Only meaningful when `batch_tracked: true`. If true, every batch record MUST carry an `expires_at` date.
`skus[].batch_scan`	enum	no	Only meaningful when `batch_tracked: true`. One of `"per_receipt"` (default) or `"per_unit"`.
`skus[].image`	object|null	no	Single product photo. Object shape `{ id, src }`. Shelfbot only re-downloads when `id` changes.
`skus[].tote_division`	int	no	Suggested tote compartment layout, encoded as `RC` (rows × cols). Shelfbot uses this internally.
`skus[].each`	object	no	Dimensions and weight of a single unit. `{ length_mm, width_mm, height_mm, weight_g }`.
`skus[].inner`	object	no	Dimensions, weight, and per-pack quantity of an inner pack.
`skus[].outer`	object	no	Dimensions, weight, and per-pack quantity of an outer pack.

Behaviour:

- For each SKU, upsert by `sku`.
- Replace the SKU's tag set with the union of `skus[].tags` and the sender's identity tag. (Tags from other senders are not affected — they live on a per-(sender, sku) basis.)
- If `image.id` differs from the cached one, schedule an async download of `image.src`.
- `tote_division`, `each`, `inner`, `outer` are stored as-is for Shelfbot's internal use; they don't trigger any side effects on update.
- Changing `batch_tracked` or `serial_tracked` on a SKU that already holds inventory is refused with `TRACKING_CHANGE_WITH_INVENTORY` — the SKU must be fully depleted first.
- This endpoint never deletes SKUs. To remove a SKU, use `/v1/sku/delete`.

Tracking invariants (Shelfbot-internal, exposed here so WMS understands refusals):

- **One batch per compartment.** A compartment holds exactly one batch at a time. Stocking a second batch into an occupied compartment is refused.
- **Serial uniqueness (per-SKU, in-stock only).** For serial-tracked SKUs, no two currently-in-stock units of the same SKU may share a serial. Different SKUs may share serial values freely. A serial that has been previously picked MAY be re-stocked (returns / RMA).
- **Batch at induct, product at pick.** For batch-tracked SKUs the batch, and the expiry where the SKU requires one, is established when stock enters a compartment: read from the carton's GS1 barcode, resolved from a carton record (*POST /v1/carton/update*), or keyed by the runner. A compartment holds one batch, so a pick inherits it: the pick scans the product barcode and no batch scan is required. Where a unit barcode does carry batch or expiry, Shelfbot may read it as a

confirmation, and a disagreement with the compartment's batch is reported as a `scan.error` exception.

- **Serials at pick.** For serial-tracked SKUs, and for lines sent with `capture_serials: true`, every unit's serial is scanned at pick. A unit without a serial barcode has its serial keyed by the operator, and `line.picked` lists it under `serials_keyed` as well as in the serials it reports.
- **No auto-decrement.** Shelfbot never auto-decrements batch- or serial-tracked stock: the quantity taken is confirmed by scan or count at the station.
- **`batch_scan.`** `per_receipt` (default): batch and expiry are recorded once per stock-in and apply to every unit stocked from that carton into the compartment; units are counted, not individually scanned for batch. `per_unit`: for products whose unit barcode carries batch and expiry; each unit is scanned as it enters the compartment and must agree with the compartment's batch or the induct is refused, and at pick the unit scan confirms the batch as well as the product.
- **Expiry is always reported.** For a SKU with `requires_expiry: true`, every batch entry Shelfbot emits or serves carries `expires_at`: in `line.picked`, the order read, the bin read, `inventory.stocked` and shipper manifests. It is the date recorded at induct for that batch, as a date only (`YYYY-MM-DD`), so it is the same value wherever that batch appears. For other batch-tracked SKUs it is present whenever it was captured.
- **Expired and blocked stock.** Stock past its expiry, or of a batch the WMS has blocked (*POST /v1/batch/update*), is excluded from strategy-driven picking and from preferred-batch fallback, including for lines already open when the block arrives: such a line draws on other batches if it can and otherwise closes short. Only a line that names the batch with `batch_policy: "required"`, typically a quarantine order, picks it. Blocked stock is refused at induct. Excluded stock stays in place and in the count until removed: by such an order, or at the station as an `inventory.adjusted` with reason `expired` or `recall`. Nothing is auto-decremented on expiry or on a block.

Per-item errors:

Code	Meaning
`INVALID_SKU`	`sku` field empty or contains disallowed characters
`IMAGE_INCOMPLETE`	`image` object present but missing `id` or `src`
`INVALID_BIN_DIVISION`	`tote_division` not in RC form with R,C ∈ 1–9
`INVALID_DIMENSIONS`	A dimension/weight field is negative, non-numeric, or `inner` / `outer` is missing required `quantity`
`INVALID_PICK_STRATEGY`	`pick_strategy` not one of `fifo`, `lifo`, `fefo`, `random`
`BARCODE_COLLISION`	`barcode` or an entry in `alt_barcodes` is already claimed by a different SKU
`INVALID_BATCH_SCAN`	`batch_scan` not one of `per_receipt`, `per_unit`
`TRACKING_CHANGE_WITH_INVENTORY`	Attempted to change `batch_tracked` or `serial_tracked` while Shelfbot still holds stock

Response:

```
{
  "ok": true,
```

```
  "message_id": "01HZ...",
  "data": {
    "results": [
      { "index": 0, "sku": "BOLT_M5_10HST_BLACK", "ok": true, "created": false,
"image_queued": true }
    ]
  }
}
```

5.2 POST /v1/sku/delete

Direction: WMS → Shelfbot **Purpose:** Delete one or more SKUs from Shelfbot. **Trigger:** WMS emits when a SKU is removed upstream.

Request:

```
{
  "skus": [
    { "sku": "BOLT_M5_10HST_BLACK", "force": false }
  ]
}
```

Fields:

Field	Type	Req	Notes
`skus[].sku`	string	yes	SKU to delete
`skus[].force`	bool	no	Default false. If true, delete even when Shelfbot holds inventory.

Behaviour:

- Look up each SKU.
- If the SKU has inventory > 0 anywhere in Shelfbot and `force != true`, refuse with `INVENTORY_NONZERO`.
- Scope guard: Shelfbot only deletes SKUs tagged with the sender's identity tag. SKUs owned by other senders, or untagged, are refused with `NOT_OWNED`.
- On success: delete the SKU, its tags, and its image references.

Per-item errors:

Code	Meaning
`INVENTORY_NONZERO`	SKU has inventory and `force` was not set
`NOT_OWNED`	SKU exists but is not tagged with the sender's identity
`UNKNOWN_SKU`	No SKU with that key

Response:

```
{
  "ok": true,
  "message_id": "01HZ...",
  "data": {
```

```
    "results": [
      { "index": 0, "sku": "BOLT_M5_10HST_BLACK", "ok": true }
    ]
  }
}
```

5.3 POST /v1/order/update

Direction: WMS → Shelfbot **Purpose:** Send a pick order to Shelfbot, or change its state (fulfil / cancel). **Trigger:** WMS emits on order creation, on cancellation, or any time the line-item set or quantities change.

Request:

```
{
  "orders": [
    {
      "id": 1739900000000,
      "name": "Job 4521",
      "state": "open",
      "priority": true,
      "line_items": [
        { "id": 900123456, "sku": "BOLT_M5_10HST_BLACK", "quantity": 4 },
        { "id": 900123457, "sku": "NUT_M5_NYL",          "quantity": 4 },
        { "id": 900123458, "sku": "MULTIMETER_FL87V",    "quantity": 1, "serials":
["FL87V-2024-0042"] },
        { "id": 900123459, "sku": "PAINT_RAL5010_1L",    "quantity": 6, "batches":
[ { "batch_number": "L240501", "expires_at": "2027-05-01" } ], "batch_policy":
"preferred" }
      ]
    }
  ]
}
```

To fulfil: send the same payload with `"state": "fulfilled"`. To cancel: send the same payload with `"state": "cancelled"`. To partially refund a line: re-send with the line's `quantity` lowered (or 0 to skip it entirely).

Fields:

Field	Type	Req	Meaning
`orders[].id`	int	yes	Stable order id chosen by the WMS. Primary key for upsert.
`orders[].name`	string	yes	Free-form display name shown in the Shelfbot UI.
`orders[].state`	enum	yes	One of `"open"`, `"fulfilled"`, `"cancelled"`.
`orders[].priority`	bool	no	Scheduling hint. `true` places the order ahead of normal orders in the pick queue; it influences sequencing and is not a completion-time guarantee. Absent on create means `false`. Absent on a later update leaves the current value unchanged.

Field	Type	Req	Meaning
`orders[].line_items[].id`	int	yes	Stable line id chosen by the WMS. Primary key for line upsert within the order.
`orders[].line_items[].sku`	string	yes	Must reference an existing SKU in Shelfbot.
`orders[].line_items[].quantity`	int	yes	Number of units to pick. Authoritative — re-sending with a lower value reduces the pick target. 0 means the line is not picked.
`orders[].line_items[].serials`	string[]	no	Only meaningful when the SKU has `serial_tracked: true`. If present, Shelfbot MUST ship these exact serials and no others (length must equal `quantity`). Used when the WMS has promised a specific unit (RMA replacement, demo unit, customer-reserved). If absent, Shelfbot picks any in-stock serials and reports them in the `line.picked` event.
`orders[].line_items[].batches`	array	no	Only meaningful when the SKU has `batch_tracked: true`. Each entry is a batch number (string) or `{ batch_number, expires_at? }`, where `expires_at` is the shelf-life date the WMS holds for that batch (`YYYY-MM-DD`). The batch number is the identity; a supplied date is a cross-check against the expiry Shelfbot recorded at induct, and a disagreement is accepted and reported as a `batch.mismatch` exception. How the named batches are used is set by `batch_policy`. If absent, Shelfbot picks per the SKU's pick strategy.
`orders[].line_items[].batch_policy`	enum	no	`"preferred"` (default when `batches` is present) or `"required"`. Preferred: pick from the named batches first; if they cannot cover the line, the remainder follows the SKU's pick strategy and the actual batches picked are reported on `line.picked`; the order is always accepted. Required: pick only from the named batches; refused at acceptance with `BATCH_NOT_AVAILABLE` if they cannot cover `quantity`, and a later shortfall closes the line short rather than substituting. Used for customer-mandated batches and for quarantine pulls of a blocked or expired batch. Ignored when `batches` is absent.

Field	Type	Req	Meaning
`orders[].line_items[].min_expires_at`	string	no	Date (`YYYY-MM-DD`). Only meaningful when the SKU has `requires_expiry: true` . Strategy-driven picking and preferred-batch fallback select only stock expiring on or after this date; a batch named in `batches` is picked regardless of the floor. A line with nothing qualifying closes with status `short` .
`orders[].line_items[].capture_serials`	bool	no	When `true` , every pick against this line requires a serial-number scan, and the picked serials are reported back to the WMS in `line.picked` events and the order read, even when the SKU is not `serial_tracked` . It does not constrain which units are picked (`serials` does that). Absent on create means `false` . Absent on a later update leaves the current value unchanged.

Behaviour:

- Upsert the order by `id` . New orders (state: `"open"`) enter the pick queue.
- Priority orders are scheduled ahead of normal orders whenever Shelfbot chooses the next bin presentations. A change to `priority` on an open order takes effect from the next scheduling decision; presentations already in flight are not interrupted.
- A line sent with `capture_serials: true` adds a serial scan to each pick: the operator scans the unit's serial number and Shelfbot reports it in the `line.picked` event. Reporting only; pick selection is unchanged.
- State transitions:
  - `open` → robot picks against the current line set.
  - `fulfilled` → Shelfbot stops picking and marks the order complete. The order's shipper labels become free for reuse (see *Shippers*).
  - `cancelled` → Shelfbot stops picking and marks the order abandoned (no fulfilment recorded). Cancellation is the WMS's alone: Shelfbot never cancels an order itself, and no `order.completed` follows a cancellation, since the WMS asked for it. Every open shipper closes with reason `order_cancelled` and its manifest, empty if nothing had been picked into it (see *Shippers*), and the order's labels become free for reuse.
- Each line item upserts by `id` . `quantity` is the live target — already-picked progress is preserved; only the remaining work changes.
- Lines previously sent that are absent from a subsequent update are left unchanged. To remove a line, re-send with `quantity: 0` .
- After update, Shelfbot re-evaluates whether the order is complete and updates its pick queue.

Per-item errors:

Code	Meaning
`UNKNOWN_SKU`	A line item references a SKU not present in Shelfbot

Code	Meaning
`EMPTY_ORDER`	`line_items` is empty on a new order
`INVALID_STATE`	`state` not one of `open`, `fulfilled`, `cancelled`
`INVALID_TRANSITION`	Attempted to move a fulfilled or cancelled order back to open
`SERIAL_NOT_AVAILABLE`	A specified serial is not currently in stock for that SKU
`SERIAL_COUNT_MISMATCH`	`serials[]` length does not equal `quantity`
`BATCH_NOT_AVAILABLE`	With `batch_policy: "required"`: a specified batch has no in-stock units, or the requested batches collectively hold fewer than `quantity` units
`INVALID_BATCH_POLICY`	`batch_policy` not one of `preferred`, `required`
`INVALID_DATE`	A batch `expires_at` or `min_expires_at` is not a `YYYY-MM-DD` date

Response:

```
{
  "ok": true,
  "message_id": "01HZ...",
  "data": {
    "results": [
      { "index": 0, "id": 1739900000000, "ok": true, "state": "open", "lines_remaining":
8 }
    ]
  }
}
```

`state` echoes Shelfbot's view after the update. `lines_remaining` is the total unpicked quantity across all lines.

5.4 POST /v1/carton/update

Direction: WMS → Shelfbot **Purpose:** Tell Shelfbot what is inside labelled cartons before they reach the trolley, so receiving becomes scan-and-confirm. **Trigger:** WMS emits when inbound carton contents become known (typically from the supplier's advance shipping notice), and on any correction.

Cartons are identified by an opaque `carton_id`: whatever unique code the carton label carries. For GS1-labelled freight that is the SSCC (Serial Shipping Container Code, application identifier 00); operations with their own labelling use their own codes, and Shelfbot treats both identically. When an operator scans a carton label at the trolley, Shelfbot resolves the contents from these records: the SKU, quantity, batch, and expiry are captured as declared rather than typed. Case labels that carry their contents in the barcode itself (GS1 application identifiers) need no carton record; this endpoint exists for cartons whose label is a bare SSCC.

Request:

```
{
  "cartons": [
    {
      "carton_id": "003123450000000011",
      "reference": "PO-88231",
```

```
      "contents": [
        {
          "sku": "PAINT_RAL5010_1L",
          "quantity": 48,
          "batch": { "batch_number": "L240901", "expires_at": "2027-03-01" }
        }
      ]
    }
  ]
}
```

Fields:

Field	Type	Req	Meaning
`cartons[].carton_id`	string	yes	The carton's label code, exactly as encoded on the label (1 to 64 characters). Typically a GS1 SSCC; any unique labelling scheme is accepted. Primary key for upsert.
`cartons[].reference`	string	no	Optional reference (purchase order, ASN or delivery id). Echoed for traceability.
`cartons[].contents[]`	array	yes	One entry per SKU in the carton; at least one entry.
`cartons[].contents[].sku`	string	yes	SKU contained.
`cartons[].contents[].quantity`	int	yes	Units of this SKU in the carton. Positive integer.
`cartons[].contents[].batch`	object	conditional	Required if the SKU is batch-tracked. Shape: `{ batch_number, expires_at?, produced_at?, supplier_batch_ref? }`.
`cartons[].contents[].serials`	string[]	conditional	For serial-tracked SKUs: the serials in the carton (length must equal `quantity`).

Behaviour (receiver / Shelfbot):

- Upsert by `carton_id`: re-sending a carton replaces its declared contents.
- Carton records are **reference data, not inventory**. Stock moves only when an `inventory.stocked` event says so; carton records carry what a carton is *declared* to contain, and Shelfbot never reconciles inventory against them.
- Carton records are **self-expiring**. Shelfbot MAY discard a record once its carton has been fully received, and discards records not referenced within 120 days. There is no carton delete endpoint; a correction is a re-send.
- An empty or over-long `carton_id`, an empty `contents` array, a non-positive quantity, or a serial list whose length does not match its quantity is refused per item with `BAD_REQUEST`. The identifier is not validated against any labelling standard: resolution at the trolley is an exact match between the scanned label and the declared code.

- The stock-in that consumes a carton reports the carton identifier back on its `inventory.stocked` event (see the `carton_id` field there), closing the loop: the WMS receives a per-carton goods-receipt confirmation.

Response: the standard envelope with per-item results, one per carton, in order.

5.5 POST /v1/batch/update

Direction: WMS → Shelfbot **Purpose:** Block or release a batch, so that a recall or a quality hold decided in the WMS takes effect inside Shelfbot. **Trigger:** WMS emits when it changes a batch's status: a supplier or regulatory recall, a quality hold, or the release of either.

Shelfbot cannot see the WMS's batch status, and its own pick strategies and preferred-batch fallback would otherwise keep offering a batch the WMS has blocked. This call closes that gap. A blocked batch is excluded from strategy-driven picking and from fallback, including for lines already open, is refused at induct if more of it arrives, and is picked only by a line that names it with `batch_policy: "required"`: the quarantine pull. Blocked and expired stock share one rule (see *Expired and blocked stock* under *POST /v1/sku/update*).

Request:

```
{
  "batches": [
    { "sku": "PAINT_RAL5010_1L", "batch_number": "L240501", "status": "blocked", "reason":
"recall", "reference": "TGA-RC-2026-0147" }
  ]
}
```

Fields:

Field	Type	Req	Meaning
`batches[].sku`	string	yes	The SKU. Batch numbers are unique only within a SKU, so both are required.
`batches[].batch_number`	string	yes	The batch. Together with `sku`, the primary key for upsert.
`batches[].status`	enum	yes	`"blocked"` or `"released"`.
`batches[].reason`	enum	no	`"recall"`, `"hold"` or `"other"`. Shown to operators and in the Portal.
`batches[].reference`	string	no	The WMS's or regulator's reference for the action. Echoed on the batch lookup.

Behaviour (receiver / Shelfbot):

- Upsert by `sku` and `batch_number`: the latest status wins, and re-sending the same status is a no-op.
- A block is accepted for a batch Shelfbot does not currently hold, so a recall that arrives before the goods do bites at receiving: the runner is refused when that batch is scanned at induct.
- `released` returns the batch to normal picking and induct. A block never deletes stock or moves it; removal is a quarantine order or a station adjustment, and the count moves only through `inventory.stocked` and `inventory.adjusted` events.

- An unknown SKU is refused per item with `UNKNOWN_SKU`; a status or reason outside the enumerations with `INVALID_STATUS` or `INVALID_REASON`.

Response: the standard envelope with per-item results, one per batch, in order.

6. Endpoints — Reading current state

Shelfbot hosts a set of GET endpoints that let the WMS read the state Shelfbot currently holds. All read endpoints are hosted by Shelfbot and called by the WMS. Reads are optional: an integration is complete without them (see the conformance statement in *Introduction*), but they support receipt checks after catalogue pushes, catalogue reconciliation, on-demand order queries triggered by events, and bin lookups.

6.1 Authentication and envelope

GET requests use the same HMAC scheme defined in *Authentication*. The request body is empty, so the signature is computed over `<timestamp>.` (the timestamp followed by a single dot, with nothing after it). All four headers are still required.

GET requests carry no request body and need not send a `Content-Type` header; responses are `application/json; charset=utf-8` as everywhere else. Read responses use the standard envelope defined in *Response envelope*, with the endpoint's payload in `data`.

6.2 GET /v1/skus

Direction: WMS → Shelfbot (read) **Purpose:** Paginated listing of the SKU codes Shelfbot currently holds for the caller.

Query parameters:

Param	Type	Req	Meaning
`updated_since`	int or string	no	Unix seconds or an ISO 8601 date. Returns only SKUs whose `updated_at` is at or after this time. `updated_at` records when Shelfbot applied the WMS's most recent change, so this filter is a receipt check: "did my recent pushes land".
`order`	enum	no	`recent` returns SKUs most-recent-first by `updated_at`. Default ordering is bytewise by SKU code, matching the sort used by the catalogue summary hash.
`limit`	int	no	Page size. Max 1000, default 500.
`cursor`	string	no	Opaque cursor from a previous page. Omit for the first page.

Response:

```
{
  "ok": true,
  "message_id": "01HZ...",
  "data": {
    "skus": [
      { "sku": "BOLT_M5_10HST_BLACK", "updated_at": "2026-08-30T09:12:44.000Z" }
    ],
    "cursor": "eyJvZmZzZXQiOjUwMH0"
```

```
  }
}
```

`cursor` is `null` on the last page. The listing is scoped to the caller's SKUs (those tagged with the sender's identity, as in *POST /v1/sku/delete*).

Deleted SKUs never appear in this listing. There are no deletion tombstones: a deletion is invisible to `updated_since`. Deletion drift is detected with the catalogue summary (*GET /v1/skus/summary*) and resolved by list comparison.

6.3 GET /v1/skus/summary

Direction: WMS → Shelfbot (read) **Purpose:** Compact summary of the caller's SKU catalogue as Shelfbot holds it, for drift detection.

Response:

```
{
  "ok": true,
  "message_id": "01HZ...",
  "data": {
    "count": 1841,
    "hash": "sha256:9f2c4e...",
    "algo": "sku-set-sha256-v1",
    "as_of": "2026-08-30T09:15:00.000Z"
  }
}
```

`data` field	Type	Meaning
`count`	int	Number of SKUs Shelfbot holds for the caller.
`hash`	string	Hash of the caller's SKU code set, computed per `algo`.
`algo`	string	Hash algorithm label. `"sku-set-sha256-v1"` in this version.
`as_of`	string	ISO 8601 UTC timestamp of the state the summary describes.

`sku-set-sha256-v1` is defined exactly as: SHA-256 over the bytewise-sorted SKU codes joined with a newline character, UTF-8 encoded, hex output prefixed `sha256:`.

The summary detects membership drift only, by design. Attribute drift is cured by re-pushing: upserts through *POST /v1/sku/update* are idempotent.

Reconciliation is WMS-driven. The WMS computes the same hash over its own SKU code set and compares it with `hash`. On mismatch, it pages through *GET /v1/skus*, diffs against its own catalogue, re-pushes missing SKUs via *POST /v1/sku/update*, and removes strays via *POST /v1/sku/delete* (subject to the `INVENTORY_NONZERO` refusal). Deletion authority stays with the WMS; Shelfbot never deletes SKUs on its own initiative.

6.4 GET /v1/sku

Direction: WMS → Shelfbot (read) **Purpose:** Read one SKU as Shelfbot currently holds it.

Query parameters: `sku=CODE` (required). The SKU code is passed as a query parameter rather than a path segment because SKU codes may contain characters hostile to URL paths.

Response:

```
{
  "ok": true,
  "message_id": "01HZ...",
  "data": {
    "sku": { "sku": "BOLT_M5_10HST_BLACK", "title": "BOLT M5 10HST Black", "...":
"remaining fields as accepted by /v1/sku/update" },
    "delivered": true,
    "updated_at": "2026-08-30T09:12:44.000Z"
  }
}
```

`data.sku` carries the same fields accepted by *POST /v1/sku/update*, with the values Shelfbot currently holds. `delivered` is `true` when the site has acknowledged the latest push for this SKU. `updated_at` is when Shelfbot applied the most recent change.

A missing `sku` parameter returns HTTP `400` with code `BAD_REQUEST`. An unknown code returns HTTP `404` with the error envelope and code `UNKNOWN_SKU`.

6.5 GET /v1/orders/{id}

Direction: WMS → Shelfbot (read) **Purpose:** Read the current execution state of an order.

Response:

```
{
  "ok": true,
  "message_id": "01HZ...",
  "data": {
    "id": 1739900000000,
    "name": "Job 4521",
    "state": "open",
    "priority": true,
    "final_state": null,
    "lines": [
      {
        "line_id": 900123456,
        "sku": "BOLT_M5_10HST_BLACK",
        "requested_quantity": 4,
        "picked_quantity": 2,
        "remaining_quantity": 2,
        "status": "partial",
        "serials": [],
        "batches": [ { "batch_number": "L240501", "quantity": 2, "expires_at":
"2027-05-01" } ]
      }
    ],
    "shippers": [
      { "shipper_seq": 1, "shipper_id": "SHP-00045-1", "status": "closed", "reason":
"full",
        "opened_at": "2026-04-23T10:11:40.000Z", "closed_at": "2026-04-23T10:14:05.000Z",
        "contents": [ { "line_id": 900123456, "sku": "BOLT_M5_10HST_BLACK", "quantity": 2,
                        "batches": [ { "batch_number": "L240501", "quantity": 2,
"expires_at": "2027-05-01" } ], "serials": [] } ] },
      { "shipper_seq": 2, "shipper_id": "SHP-00045-2", "status": "open", "reason": null,
        "opened_at": "2026-04-23T10:14:20.000Z", "closed_at": null, "contents": [] }
    ],
```

```
    "accepted_at": "2026-04-23T10:11:02.000Z",
    "completed_at": null,
    "as_of": "2026-04-23T10:16:00.000Z"
  }
}
```

`state` mirrors the order state enum in *POST /v1/order/update*. `priority` echoes the current value of the flag set there (`false` when never set). `final_state` is `null` until the order reaches a terminal state, then carries the same value as the `order.completed` event's `final_state`. Per-line `status` uses the same enum as `line.picked`. `serials` and `batches` accumulate what has been picked so far, each batch entry carrying its `expires_at` per *Expiry is always reported*; the line's `batch_policy` and `min_expires_at` are echoed as sent. `shippers` lists every shipper the order has used, in `shipper_seq` order: its label, `status` (`open` or `closed`), the close `reason` (`full`, `order_complete`, `order_cancelled`, or `null` while open), timestamps, and `contents`, which grows as units are put in and is final once the shipper closes, matching its `shipper.closed` event. An order picked into a single shipper shows one entry (see *Shippers*).

This endpoint is served from state Shelfbot maintains at its edge, materialised from the event stream. It may lag live activity by moments; a read issued after an event has been received is guaranteed not to be older than that event. When the site is offline, the endpoint keeps answering with last-known state, and `as_of` shows how fresh that state is. Completed orders remain readable for 90 days after completion.

An unknown id returns HTTP `404` with the error envelope and code `UNKNOWN_ORDER`.

6.6 GET /v1/bins/{bin_id}

Direction: WMS → Shelfbot (read) **Purpose:** Read the current contents of a bin.

Response:

```
{
  "ok": true,
  "message_id": "01HZ...",
  "data": {
    "bin_id": 40213,
    "contents": [
      {
        "sku": "PAINT_RAL5010_1L",
        "quantity": 6,
        "batches": [ { "batch_number": "L240501", "quantity": 6, "expires_at":
"2027-05-01", "status": "active" } ],
        "serials": []
      }
    ],
    "as_of": "2026-04-23T10:16:00.000Z"
  }
}
```

Each batch entry carries its `status`: `active`, `blocked` (*POST /v1/batch/update*) or `expired`. Bins are Shelfbot's physical domain; the WMS may read bin state but never commands bin placement.

An unknown bin id returns HTTP `404` with the error envelope and code `UNKNOWN_BIN`.

6.7 GET /v1/shippers/{shipper_id}

Direction: WMS → Shelfbot (read) **Purpose:** Resolve a shipper label to the order it currently holds, with its contents. Typically called when a packing station scans a shipper and the WMS wants to confirm whose order is inside.

Response:

```
{
  "ok": true,
  "message_id": "01HZ...",
  "data": {
    "shipper_id": "SHP-00045-1",
    "order_id": 1739900000000,
    "external_id": null,
    "shipper_seq": 1,
    "status": "closed",
    "reason": "full",
    "opened_at": "2026-04-23T10:11:40.000Z",
    "closed_at": "2026-04-23T10:14:05.000Z",
    "contents": [
      { "line_id": 900123456, "sku": "BOLT_M5_10HST_BLACK", "quantity": 2,
        "batches": [ { "batch_number": "L240501", "quantity": 2, "expires_at":
"2027-05-01" } ], "serials": [] }
    ],
    "as_of": "2026-04-23T10:16:00.000Z"
  }
}
```

The `shipper_id` in the path is the label code exactly as printed, percent-encoded where it contains characters outside the unreserved URL set. Because labels are reused, the response is the label's current binding, which persists until the WMS closes the order (see *Shippers*). After that the label is free, and the lookup returns HTTP `404` with the error envelope and code `UNKNOWN_SHIPPER` until the label is inducted again. The endpoint is served from the same edge-materialised state as *GET /v1/orders/{id}*, with the same freshness rules.

6.8 GET /v1/batches

Direction: WMS → Shelfbot (read) **Purpose:** Locate a batch: how much Shelfbot holds, in which bins, with its expiry and status. The locate and reconcile steps of a recall, before the quarantine pull and after it.

Query parameters: `sku=CODE` (required) and `batch_number=...` (optional). Without `batch_number` the response lists every batch Shelfbot holds or has a status for under that SKU. Both are query parameters because batch numbers, like SKU codes, may contain characters hostile to URL paths.

Response:

```
{
  "ok": true,
  "message_id": "01HZ...",
  "data": {
    "sku": "PAINT_RAL5010_1L",
    "batches": [
      {
        "batch_number": "L240501",
```

```
        "status": "blocked",
        "reason": "recall",
        "reference": "TGA-RC-2026-0147",
        "expires_at": "2027-05-01",
        "on_hand": 14,
        "bins": [ { "bin_id": 40213, "quantity": 6 }, { "bin_id": 40391, "quantity": 8 } ]
      }
    ],
    "as_of": "2026-04-23T10:16:00.000Z"
  }
}
```

`status` is `active`, `blocked` or `expired`; `reason` and `reference` are present for a blocked batch, as sent on *POST /v1/batch/update*. `on_hand` is the sum over `bins`. A batch that has been blocked before any of it arrived appears with `on_hand` 0 and no bins. After a quarantine pull the batch reads with `on_hand` 0, which is the reconciliation evidence; a non-zero figure after the pull is stock the pull missed.

Served from the same edge-materialised state as the bin read, with the same freshness rules. An unknown SKU returns HTTP `404` with the error envelope and code `UNKNOWN_SKU`; a `batch_number` Shelfbot has never held or been sent a status for returns `404` with code `UNKNOWN_BATCH`.

7. Endpoints — Shelfbot → WMS

Shelfbot → WMS endpoints reuse the auth, batching, idempotency, and retry rules defined in *Protocol*. The HMAC key id in this direction identifies Shelfbot to the WMS. Acknowledgement in this direction is simpler than the full response envelope; see *Acknowledgement* below.

7.1 Generation and delivery

Shelfbot creates events as operational transactions are committed. Delivery is asynchronous: for efficiency, multiple queued events may be delivered in a single request. Batching for transport never alters each event's `occurred_at` or semantics.

For example, a single POST may carry three events: two `line.picked` events from bin presentations thirty seconds apart, and one `inventory.stocked` event. Each keeps its own `occurred_at`:

```
{
  "events": [
    { "event_id": "01J6W0A0A0A0A0A0A0A0A0A0A1", "occurred_at": "2026-08-30T10:15:00.000Z",
      "type": "line.picked", "order_id": 1739900000000, "external_id": null,
      "data": { "line_id": 900123456, "sku": "BOLT_M5_10HST_BLACK", "requested_quantity":
8,
                "picked_quantity": 4, "status": "partial", "bin_id": 40211 } },
    { "event_id": "01J6W0A0A0A0A0A0A0A0A0A0A2", "occurred_at": "2026-08-30T10:15:30.000Z",
      "type": "line.picked", "order_id": 1739900000000, "external_id": null,
      "data": { "line_id": 900123456, "sku": "BOLT_M5_10HST_BLACK", "requested_quantity":
8,
                "picked_quantity": 4, "status": "complete", "bin_id": 40212 } },
    { "event_id": "01J6W0A0A0A0A0A0A0A0A0A0A3", "occurred_at": "2026-08-30T10:16:05.000Z",
      "type": "inventory.stocked",
      "data": { "sku": "NUT_M5_NYL", "delta": 200, "new_count": 417, "operator":
"user-17", "bin_id": 40213 } }
```

```
  ]
}
```

A committed transaction may emit multiple events, one per business fact: a pick, a stock-in, and an adjustment can share one bin visit. Corrections made before a transaction commits are not events. Net removals outside a pick are reported as `inventory.adjusted`. A net-zero change emits nothing.

The event vocabulary maps onto the WMS's own documents: `inventory.stocked` corresponds to the WMS's goods-receipt document; `inventory.adjusted` corresponds to its stock-correction document, with reason codes; picks are neither. Picks arrive only as order events and are never double-counted in inventory events.

7.2 Acknowledgement

Requests into Shelfbot (the WMS to Shelfbot endpoints and the read endpoints) return the full response envelope with per-item results, exactly as defined in *Protocol*. That contract is unchanged.

Event deliveries from Shelfbot to the WMS are acknowledged differently:

- Any HTTP 2xx response acknowledges the whole batch.
- No response body is required, and any body that is returned is ignored.
- A non-2xx response or a timeout follows the retry policy in *Retry policy (sender)*.
- Receivers SHOULD acknowledge promptly and process asynchronously.

A receiver that does not recognise an individual event (an unknown `order_id`, an unknown SKU, an unrecognised type) MUST still acknowledge the batch with 2xx and SHOULD log the event. A batch is never failed over one unrecognised event.

Receivers MUST acknowledge and ignore event types they do not process. New event types may be added in future versions of this specification, and ignoring them is always safe.

7.3 POST /v1/order/event

Direction: Shelfbot → WMS **Purpose:** Report progress and terminal state of an order back to the WMS: line progress, line closure, the shippers it is picked into, order completion, failure. **Trigger:** Shelfbot emits as picking progresses. A line may emit multiple `line.picked` events (see multi-event semantics below). One `order.completed` event fires when the order reaches a terminal state. Shipper events fire when a shipper is inducted for the order and when it closes.

Request:

```
{
  "events": [
    {
      "event_id": "01HZQX5N4F8R7Y3K0M2D9V6BAE",
      "occurred_at": "2026-04-23T10:15:30.123Z",
      "type": "line.picked",
      "order_id": 1739900000000,
      "external_id": null,
      "data": {
        "line_id": 900123456,
        "sku": "BOLT_M5_10HST_BLACK",
        "requested_quantity": 4,
        "picked_quantity": 4,
```

```
        "status": "complete",
        "bin_id": 40211,
        "shipper_id": "SHP-00045-1",
        "shipper_seq": 1,
        "operator": "user-17"
      }
    },
    {
      "event_id": "01HZQX5N7Y2K8M3D0V6BAERFGH",
      "occurred_at": "2026-04-23T10:18:42.555Z",
      "type": "order.completed",
      "order_id": 1739900000000,
      "external_id": null,
      "data": {
        "final_state": "fulfilled",
        "units_picked": 8,
        "units_requested": 8,
        "duration_ms": 192432
      }
    }
  ]
}
```


Common event fields:

Field	Type	Req	Meaning
`events[].event_id`	string	yes	ULID or UUID, unique per event. WMS dedupes on this for 24h (in addition to batch-level `X-Shelfbot-Message-Id`).
`events[].occurred_at`	string	yes	ISO 8601 UTC timestamp of when the event happened on Shelfbot. May predate the request by hours if Shelfbot was offline.
`events[].type`	enum	yes	One of the event types listed below.
`events[].order_id`	int	yes	The `order_id` originally sent by the WMS in `/v1/order/update`.
`events[].external_id`	int|string|null	no	Echoed from the order's `external_id` if the WMS supplied one. Null if not.
`events[].data`	object	yes	Type-specific payload (see per-type sections).

Event types

`line.picked`

Emitted when a committed bin presentation completes against a line. The grain of this event is the committed bin presentation, not an individual scan. Most lines need one presentation, so most lines emit exactly one `line.picked` event carrying the full commanded quantity. A line whose stock spans multiple bins emits one event per presentation: each carries the incremental `picked_quantity` with status `"partial"`, and the last carries status `"complete"`. A short pick followed by a top-up from another bin follows the same pattern.

Multi-event semantics:

- `picked_quantity` on each event is **incremental**: the units removed in this specific presentation, not the cumulative total.
- The WMS sums `picked_quantity` across events for a given `line_id` to get cumulative progress.
- `status` describes the line's state at the moment this event closes:
  - `"complete"`: this presentation brought the line to fully picked (cumulative sum equals `requested_quantity`). No more `line.picked` events for this line.
  - `"partial"`: this presentation contributed to the line, but the line is not yet fully picked and further `line.picked` events are expected (the line's remaining stock is in another bin, or the operator has routed the order to an issues location for later top-up).
  - `"short"`: this presentation closed the line with less than `requested_quantity`. No more events expected; line is final.
  - `"missing"` — line closed with zero picked (SKU not findable). No more events expected.
  - `"manual"`: line satisfied outside the normal pick flow (operator override at HMI); `picked_quantity` reflects what the operator recorded.

Un-picks and corrections: corrections made before a transaction commits never become events; a unit scanned and then returned during a presentation is invisible to the WMS. Post-commit discrepancies surface as `inventory.adjusted` events (reason codes `mis_picked`, `missing`). `order.completed` always carries the final authoritative totals.

Short-pick with recount: when a presentation ends with the bin count wrong (Shelfbot expected 4, the operator found 2), the HMI requires the operator to confirm the bin's remaining count before releasing the bin. This generates:

1. A `line.picked` event with the units actually removed and status reflecting line state.
2. A separate `inventory.adjusted` event (*POST /v1/inventory/event*) reporting the count correction, with `reason.code: "missing"`.

The two events may be emitted in the same batch.

`data` field	Type	Req	Meaning
`line_id`	int	yes	The `line_id` originally sent by the WMS.
`sku`	string	yes	SKU picked (or attempted).
`requested_quantity`	int	yes	Quantity the WMS asked for (current value of `quantity` on the line).
`picked_quantity`	int	yes	Units picked in **this presentation**. Incremental; the WMS sums across events.
`status`	enum	yes	See multi-event semantics above.
`bin_id`	int	yes	The bin presented for this event.
`shipper_id`	string	conditional	Label of the shipper the units were put into (see *Shippers*). Present when `picked_quantity` is greater than zero.
`shipper_seq`	int	conditional	Sequence of that shipper within the order. Present with `shipper_id`.

`data` field	Type	Req	Meaning
`operator`	string	no	Operator id if a human picked or marked manual; omitted for fully-robotic picks.
`batches`	array	conditional	Required if the SKU is batch-tracked. Breakdown of which batch(es) the units came from in this presentation. Each entry: `{ batch_number, quantity, expires_at, serials?[] }`; `expires_at` is required when the SKU has `requires_expiry: true` and present otherwise whenever it was captured (see *Expiry is always reported*). `serials[]` required when the SKU is both batch- and serial-tracked.
`serials`	string[]	conditional	Required if the line calls for serial reporting (the SKU is serial-tracked, or the line was sent with `capture_serials: true`) and the SKU is not batch-tracked. Exact serials picked in this presentation (length must equal `picked_quantity`). When both batch and serial are tracked, serials live inside `batches[].serials`.
`serials_keyed`	string[]	no	Serials the operator keyed rather than scanned, a subset of the serials reported on this event wherever they appear (see *Serials at pick*).

Note: `bin_id` identifies the bin presented. Compartment structure within a bin remains internal to Shelfbot; the WMS may read bin state via *GET /v1/bins/{bin_id}* but never commands bin placement.

`order.completed`

Emitted exactly once per order when it reaches a terminal state.

`data` field	Type	Req	Meaning
`final_state`	enum	yes	`"fulfilled"` (every line complete), `"partially_fulfilled"` (at least one short or missing, at least one complete), `"failed"` (no lines picked at all).
`units_picked`	int	yes	Sum of cumulative `picked_quantity` across all lines.
`units_requested`	int	yes	Sum of `requested_quantity` across all lines.
`duration_ms`	int	no	Wall time from order acceptance to completion.

`shipper.opened`

Emitted when the runner inducts a shipper for the order: the scan that binds the label to a trolley position and to the order. The order's first shipper opens when the order is first bound to the trolley; a further shipper opens after the previous one closed full.

`data` field	Type	Req	Meaning
`shipper_id`	string	yes	The label code exactly as scanned (1 to 64 characters). Opaque to Shelfbot.

`data` field	Type	Req	Meaning
`shipper_seq`	int	yes	1 for the order's first shipper, 2 for the next, and so on.

`shipper.closed`

Emitted when a shipper stops receiving picks for the order: Shelfbot tagged it full and the runner is swapping it out, the order reached a terminal state (its last shipper closes with it), or the WMS cancelled the order. Every shipper that opened closes exactly once, whatever the reason, so the WMS can pair each `shipper.opened` with its `shipper.closed` and always knows when a label is free. `contents` is the shipper's manifest and is final; it is empty when nothing was put into the shipper before it closed.

`data` field	Type	Req	Meaning
`shipper_id`	string	yes	The label code, as on `shipper.opened`.
`shipper_seq`	int	yes	As on `shipper.opened`.
`reason`	enum	yes	`"full"`, `"order_complete"` or `"order_cancelled"`.
`contents`	array	yes	What is in the shipper: one entry per line with units in it, `{ line_id, sku, quantity, batches?[], serials?[] }`, with `batches` and `serials` shaped and required as on `line.picked`. Empty when nothing was put into the shipper before it closed.

The order is not complete when a shipper closes full: `order.completed` still fires once, when the last line closes. Between the two, the WMS knows which lines are in which shipper and runs its own process, dispatching each shipper as it arrives or holding for the order, as it sees fit. A shipper's label stays bound to the order until the WMS closes the order (see *Shippers*).

Behaviour (receiver / WMS):

- Verify auth, dedupe on `X-Shelfbot-Message-Id` (batch) and per-event on `event_id`.
- Apply each event in `events[]` order. Ordering is best-effort — WMS MUST tolerate out-of-order delivery. Use `occurred_at` to reconstruct the true order if needed.
- For `line.picked`, maintain a running sum of `picked_quantity` per `line_id`. Latest `status` wins for line state.
- For `shipper.closed`, treat `contents` as the shipper's final manifest. `shipper.opened` may be ignored by a WMS that only wants manifests.
- Unknown `order_id`: acknowledge the batch with 2xx, log the event, and continue. Never fail a batch over one unrecognised event.

Acknowledgement: any HTTP 2xx response acknowledges the whole batch; no response body is required and any body is ignored. See *Acknowledgement*.

7.4 POST /v1/inventory/event

Direction: Shelfbot → WMS **Purpose:** Report inventory changes that occur outside the order-pick flow — stock-in, manual count corrections, and short-pick recounts. **Trigger:** Shelfbot emits whenever physical stock changes for a reason other than fulfilling an order line. (Pick-driven stock decrements are implied by `line.picked` events in *POST /v1/order/event* and are NOT re-emitted here.)

Request:

```
{
  "events": [
    {
      "event_id": "01HZQX6P8M2N4R7K0V3D9Y6BAE",
      "occurred_at": "2026-04-23T11:02:14.000Z",
      "type": "inventory.stocked",
      "data": {
        "sku": "BOLT_M5_10HST_BLACK",
        "delta": 500,
        "new_count": 540,
        "bin_id": 40213,
        "operator": "user-17",
        "reference": "PO-78421",
        "batch": { "batch_number": "L240423", "expires_at": "2027-04-23" },
        "serials": []
      }
    },
    {
      "event_id": "01HZQX6P9N3K8M2D0V6BAEFGHJ",
      "occurred_at": "2026-04-23T11:05:30.000Z",
      "type": "inventory.adjusted",
      "data": {
        "sku": "NUT_M5_NYL",
        "delta": -3,
        "new_count": 217,
        "bin_id": 40214,
        "operator": "user-17",
        "reason": { "code": "missing" },
        "batch_number": "L240301",
        "serials_affected": []
      }
    }
  ]
}
```


Important semantic note: `new_count` is Shelfbot's new **total stock of the SKU** across all of its managed storage (or, for batch-tracked SKUs, the new total for the specific batch). It is not a per-tote or per-compartment count. The WMS can use this as a cross-check against its own running total; if they diverge, something is wrong.

Event types

`inventory.stocked`

A receiving / restocking action — units added to Shelfbot's managed stock from outside (delivery, transfer in).

One event is emitted per bin return: the bin coming back is the committed transaction, and `delta` is the net units added during that presentation.

`data` field	Type	Req	Meaning
`sku`	string	yes	SKU added.
`delta`	int	yes	Net units added during this presentation. Always positive for `stocked`.
`new_count`	int	yes	Shelfbot's new total for this SKU (or batch, for batch-tracked SKUs) after the change.

`data` field	Type	Req	Meaning
`bin_id`	int	yes	The bin the units were stocked into.
`operator`	string	yes	Operator id who did the stock-in.
`reference`	string	no	Optional reference (purchase order, transfer id, etc.).
`carton_id`	string	no	The carton this stock-in came from, when the operator received by carton scan (see *POST /v1/carton/update*). A per-carton goods-receipt confirmation.
`batch`	object	conditional	Required if the SKU is batch-tracked. Shape: `{ batch_number, expires_at?, produced_at?, supplier_batch_ref? }`. `expires_at` required when `requires_expiry: true`.
`serials`	string[]	conditional	Required if the SKU is serial-tracked. Array of serials, one per unit (length must equal `delta`).

`inventory.adjusted`

A stock correction — Shelfbot's count for a SKU (or batch) changed for a reason other than a receipt or a pick. Fires in two scenarios:

1. **Short-pick recount.** Operator found fewer units in a tote than expected; the recount at HMI generates this event alongside the accompanying `line.picked`.
2. **Standalone adjustments.** Cycle counts, damage write-offs, recalls, etc., initiated at a Shelfbot station.

`delta` may be positive (found extra) or negative (missing, damaged, expired, etc.). `delta: 0` is permitted to record that a count was performed and confirmed correct.

`data` field	Type	Req	Meaning
`sku`	string	yes	SKU adjusted.
`delta`	int	yes	Signed difference between new count and previous count.
`new_count`	int	yes	Shelfbot's new total for this SKU (or batch) after the adjustment.
`bin_id`	int	yes	The bin whose contents were adjusted.
`operator`	string	yes	Operator id — required for audit trail.
`reason`	object	yes	Structured reason. Shape: `{ code: enum, note?: string }`. See reason codes below. `note` is required when `code` is `"other"`, optional otherwise.
`batch_number`	string	conditional	Required if the SKU is batch-tracked — identifies which batch was adjusted.
`serials_affected`	string[]	no	For serial-tracked SKUs: the specific serials added (positive `delta`) or removed (negative `delta`) from stock. If the operator couldn't identify individual serials during a count, omit; Shelfbot will flag unreconciled serials via an `integrity` exception (*POST /v1/exception*).

Reason codes:

Code	Meaning
`missing`	Expected units weren't there, no further explanation (routine short-pick correction, cycle count discrepancy).
`damaged_removed`	Units were there but unusable; operator removed them from stock.
`found_extra`	More units than Shelfbot expected (positive `delta`).
`expired`	Units past expiry date, removed from pickable stock.
`quality_hold`	Units pulled for QA inspection; may return later via `inventory.stocked`.
`cycle_count`	Routine count reconciliation with no specific cause; use when a discrepancy is found during a scheduled count and the operator doesn't know why.
`recall`	Supplier or regulatory recall; units pulled from pickable stock. Typical for batch-tracked SKUs.
`mis_picked`	Correcting a previous mis-pick (operator took the wrong SKU). Usually paired — one negative adjustment on the SKU that was over-picked, one positive on the SKU that was under-picked.
`returned_to_supplier`	Stock sent back to supplier (wrong item, over-shipment).
`sample_taken`	Units removed for QA sampling, testing, or customer samples; not returning to stock.
`transfer_out`	Units moved to another location outside Shelfbot's managed stock.
`transfer_in`	Units moved in from another location outside Shelfbot's managed stock.
`other`	Free-form reason — `note` field required and must describe the cause.

Behaviour (receiver / WMS):

- Verify auth, dedupe on `X-Shelfbot-Message-Id` and per-event on `event_id`.
- Apply each event to the WMS's view of inventory. The WMS holds the authoritative count; `new_count` is informational for cross-check.
- Out-of-order delivery is possible. Use `occurred_at` to sequence; if an older event arrives after a newer one, WMS SHOULD log the late event but NOT regress its count.
- Unknown SKU: acknowledge the batch with 2xx, log, and consider triggering a SKU resync. Never fail a batch over one unrecognised event.

Acknowledgement: as for *POST /v1/order/event*.

7.5 POST /v1/exception

Direction: Shelfbot → WMS **Purpose:** Report abnormal conditions that fall outside the normal pick / stock / adjust flow — hardware faults, scanner errors, integrity violations, operator overrides. These are informational / audit events; the WMS is NOT expected to take corrective action. **Trigger:** Shelfbot emits when an unexpected condition occurs that the WMS might want to log, alert on, or surface in dashboards.

Request:

```
{
  "events": [
    {
      "event_id": "01HZQX7R3K9M2N5D0V6BAEFGHJ",
      "occurred_at": "2026-04-23T12:14:05.000Z",
      "type": "robot.fault",
      "severity": "error",
      "context": { "robot_id": "SB-2" },
      "data": {
        "code": "MOTOR_STALL",
        "message": "Y-axis motor stalled at position 3200mm",
        "recoverable": true
      }
    },
    {
      "event_id": "01HZQX7R5N3K8M2D0V6BAEFGHJ",
      "occurred_at": "2026-04-23T12:16:44.000Z",
      "type": "scan.error",
      "severity": "info",
      "context": { "operator": "user-17" },
      "data": {
        "barcode_read": "5099999999999",
        "reason": "unknown_barcode"
      }
    }
  ]
}
```

Common event fields:

Field	Type	Req	Meaning
`events[].event_id`	string	yes	ULID or UUID. Per-event dedupe.
`events[].occurred_at`	string	yes	ISO 8601 UTC.
`events[].type`	enum	yes	One of the exception types below.
`events[].severity`	enum	yes	`"info"`, `"warning"`, `"error"`, `"critical"`. Lets the WMS route to appropriate dashboards / alerting.
`events[].context`	object	no	Scoping info. Allowed fields: `robot_id`, `sku`, `order_id`, `line_id`, `operator`. Shelfbot-internal identifiers (tote ids, compartment indices, station ids) are never exposed.
`events[].data`	object	yes	Type-specific payload.

Exception types

`robot.fault`

Hardware or firmware fault on a Shelfbot robot.

`data` field	Type	Req	Meaning
`code`	string	yes	Short machine-readable code, e.g. `"MOTOR_STALL"`, `"LIMIT_SWITCH"`, `"COMMS_LOST"`, `"ESTOP_TRIGGERED"`, `"THERMAL"`. Deployments may extend.

`data` field	Type	Req	Meaning
`message`	string	yes	Human-readable description.
`recoverable`	bool	yes	true if Shelfbot expects to auto-recover; false if it needs maintenance.

`scan.error`

A scan could not be resolved or was invalid. Not all scan errors emit events — only those the WMS might care about (e.g. repeated unknown barcodes may indicate a SKU sync issue).

`data` field	Type	Req	Meaning
`barcode_read`	string	yes	The raw barcode value that was scanned.
`reason`	enum	yes	`"unknown_barcode"`, `"damaged_barcode"`, `"wrong_context"`.

`batch.mismatch`

The WMS supplied a shelf-life date for a batch on an order line (*POST /v1/order/update*) that disagrees with the expiry Shelfbot recorded for that batch at induct. The order is accepted and picked by batch number; this exception tells both sides to reconcile their batch data.

`data` field	Type	Req	Meaning
`order_id`	int	yes	The order carrying the line.
`line_id`	int	yes	The line.
`sku`	string	yes	The SKU.
`batch_number`	string	yes	The batch in question.
`wms_expires_at`	string	yes	The date the WMS supplied.
`shelfbot_expires_at`	string	yes	The date Shelfbot holds from induct.

`integrity`

Shelfbot detected an internal consistency issue — count went negative, two batches found in one compartment, orphaned serial, etc. These indicate a bug or data corruption; the WMS SHOULD alert loudly.

`data` field	Type	Req	Meaning
`code`	string	yes	e.g. `"NEGATIVE_COUNT"`, `"MIXED_BATCH_COMPARTMENT"`, `"ORPHAN_SERIAL"`, `"COMPARTMENT_OVERFILL"`.
`message`	string	yes	Human-readable description.
`auto_corrected`	bool	yes	true if Shelfbot made a safe correction (e.g. clamped count to 0); false if the system is left in an inconsistent state pending human review.

`override`

An operator bypassed a Shelfbot refusal (typically after an HMI alert). Always emit so the WMS has an audit record of who overrode what.

`data` field	Type	Req	Meaning
`override_of`	string	yes	The error code that was overridden (e.g. `"BATCH_MISMATCH"`, `"SERIAL_COLLISION"`, `"COUNT_MISMATCH"`).
`operator`	string	yes	Operator id who authorised the override.
`reason`	string	yes	Free-form reason supplied by the operator.
`auth_level`	enum	no	`"operator"`, `"supervisor"`, `"admin"` — if the deployment uses tiered authorisation.

Behaviour (receiver / WMS):

- Verify auth, dedupe on `X-Shelfbot-Message-Id` (batch) and per-event on `event_id`.
- Log / store each event. Route by severity to appropriate channels (info → audit log; warning → dashboard; error/critical → alerting).
- No corrective action is expected from the WMS.
- Unknown type: acknowledge the batch with 2xx and log the unknown type. Ignoring exception types the WMS does not process is always safe and allows Shelfbot to add new exception types without breaking older WMSes.

Acknowledgement: as for *POST /v1/order/event*.

8. Notes & Responsibilities

8.1 WMS Integrator responsibilities

- **Credential custody** — safeguarding the HMAC shared secret. Never committing secrets to version control.
- **SKU sync** — keeping Shelfbot's SKU catalogue in sync with the WMS master data.
- **Order lifecycle** — creating orders via `/v1/order/update` with `state: "open"`, and closing every order with `fulfilled` or `cancelled` once the WMS has finished with it. A shipper label stays bound to its order until then, so an order left open holds its shippers (see *Shippers*).
- **Batch status:** blocking a recalled or held batch through `/v1/batch/update` as soon as the WMS blocks it, and releasing it when the hold ends. Shelfbot cannot see the WMS's batch status and will otherwise keep offering the batch.
- **Event receipt** — hosting the Shelfbot → WMS endpoints (*Endpoints — Shelfbot → WMS*) with correct auth verification, idempotency handling, and tolerance for out-of-order delivery.
- **Cumulative pick tracking** — summing `picked_quantity` across `line.picked` events per `line_id` to derive cumulative progress.
- **Inventory reconciliation** — treating inbound `inventory.stocked` and `inventory.adjusted` events as authoritative signals that the WMS's own stock count needs to move by `delta`. Using `new_count` as a cross-check.
- **Failure handling** — honouring the retry policy in *Retry policy (sender)*.
- **Audit retention** — retaining event records for the customer's regulatory environment.

8.2 Shelfbot responsibilities

- **Physical-state integrity** — keeping tote-level inventory consistent with physical reality. Refusing any request that would violate a documented invariant.
- **HMI operator flows** — surfacing invariant violations at the station and halting local flow until resolved. Forcing tote recount after short picks.
- **Scanning** — owning all barcode scanning at stations; validating scans against current tote/order state before emitting events.
- **Event emission** — emitting `/v1/order/event`, `/v1/inventory/event`, and `/v1/exception` events as the authoritative record of what happened on the picking floor.
- **Shipper assignment:** deciding which shipper each order is picked into and when a shipper is full, and reporting it through the shipper events and reads.
- **Honouring batch status:** excluding blocked and expired batches from picking and refusing them at induct, and serving the batch lookup so a recall can be located and reconciled.
- **Operational telemetry:** Shelfbot retains operational telemetry and performance data for monitoring, support, and reporting, surfaced through Shelfbot's own dashboards. The WMS is not required or expected to derive Shelfbot equipment KPIs from transactional events; `duration_ms` on `order.completed` remains available for WMSes that want it.
- **Auth enforcement** — verifying HMAC signatures, timestamps, and (if configured) IP allowlists.
- **Backward compatibility** — within a major version, preserving existing field semantics and only introducing additive changes.

8.3 Shared responsibilities

Before go-live, both parties must agree on:

- HMAC key exchange (Shelfbot issues, WMS stores securely).
- Webhook base URLs for each direction.
- Optional IP allowlist contents.
- Expected batch/serial tracking per SKU class, any `requires_expiry` policy, the `batch_policy` the WMS sends by default, and how cartons bound for Shelfbot carry batch and expiry (GS1 content barcodes, carton records, or keyed at induct).
- Retention period for events and audit records on each side.

Any deviation from this specification must be agreed in writing between the WMS operator and Shelfbot prior to deployment.

9. Revision History

Version	Date	Change
3.1	19/08/2026	Published to Web Site
3.2	01/09/2026	Batch size errors return 400; webhook acknowledgement simplified to any 2xx; transaction-grain event semantics with bin_id; read endpoints for SKUs, orders, and bins; catalogue summary for reconciliation; integration profiles; carton feed (SSCC) for receiving

Version	Date	Change
3.3	10/09/2026	Order priority flag: optional `priority` on order create and update, honoured in pick-queue sequencing and echoed by the order read; line-level `capture_serials`: an order line can require the picked serial numbers to be scanned and reported back to the WMS; shippers: `shipper.opened` and `shipper.closed` events carrying the shipper manifest, shipper on `line.picked`, shippers on the order read and a lookup by label; the WMS closes every order with `fulfilled` or `cancelled` to release its shipper labels; the never-emitted `order.cancelled` event removed: cancellation is the WMS's alone; batch handling aligned to receipt-led practice: `batch_policy` (preferred with strategy fallback, or required), a shelf-life date and `min_expires_at` on the line, `fefo` pick strategy (default for expiry-tracked SKUs), batch established at induct with the product scanned at pick, `batch_scan` defined, expiry always reported for expiry-tracked SKUs, expired stock rule, keyed serials marked, `batch.mismatch` exception; recall loop: `POST /v1/batch/update` blocks or releases a batch (excluded from picking including open lines, refused at induct, pulled only by a required-batch order), `GET /v1/batches` locates a batch with on hand per bin, batch status on the bin read