OTEP — Open Tracking Event Protocol

Status: Draft v0.1 · An open, vendor-neutral specification · License: open (see §10)

OTEP is an open, vendor-neutral protocol for the lifecycle of any trackable subject. It defines one event model and one status vocabulary so that tracking events from any source — own-fleet delivery, third-party couriers, carrier labels and beyond — can be exchanged, understood and projected to international standards without re-integrating for every party.

This document is the specification: the structure, the fields, the status tables, the state machine, the external-standard mappings, and the conformance rules for building a compliant implementation. It is implementation-agnostic — it describes the protocol, not any particular vendor's internals. RFC-2119 keywords (MUST / SHOULD / MAY) are normative.

1. What OTEP solves

Tracking is fragmented: every carrier names fields and status codes differently, every delivery channel reports in its own shape, and connecting to global standards means re-integrating again and again. OTEP gives producers and consumers a single common language: a producer emits OTEP events once, and every OTEP consumer understands them and can project them to the standard it needs.

2. Design principles

1. Event-sourced. The event timeline is the source of truth; "current status" is always a projection — the status of the latest event.
2. What / When / Where / Why. Every event is shaped around these four dimensions — the same dimensions shared by GS1 EPCIS, IATA ONE Record and UN/CEFACT — so those standards are output projections of an OTEP event rather than parallel models.
3. No-list sources are not a blocker. A source that exposes only a current status (no history) is handled by synthesizing one event per observed change (§5).
4. Additive. OTEP is exposed alongside any existing tracking API; adopting it never requires a breaking change to what consumers already use.

3. The OTEP timeline

A timeline is an envelope carrying the tracked subject and an ordered list of events.

```
{
  "otep_version": "0.1",
  "profile": "parcel",                  // domain profile (§8)
  "subject": {
    "tracking_number": "SR123...",      // ?1 identifier required
    "order_id": 12345,                  // optional
    "package_id": 67890,                // optional
    "external_tracking_number": "1Z...",// optional
    "gs1_sscc": "00...",                // optional — enables EPCIS epcList
    "piece_id": "..."                   // optional — enables ONE Record linkage
  },
  "current_status": "delivered",        // projection of the latest event
  "delivered": true,
  "events": [ /* OTEP events, §4 */ ]
}
```

The OTEP event

```
{
  // WHAT — carried once at the timeline level (subject above)

  // WHEN
  "occurred_at": "2026-06-10T09:30:00-04:00",  // event instant, ISO-8601 with offset
  "recorded_at": "2026-06-10T09:45:23-04:00",  // ingestion instant (optional)
  "time_type": "actual",                        // actual | estimated | scheduled
```

```
  // WHERE
  "location": {
    "name": "Toronto Hub",
    "code": "YYZ-2",
    "gln": "0614141000005",                      // GS1 GLN ? EPCIS SGLN
    "lat": 43.6777, "lng": -79.6248,
    "country": "CA"                              // ISO 3166-1 alpha-2
  },

  // WHY / WHAT HAPPENED
  "status_code": "out_for_delivery",             // an OTEP status code (§4)
  "phase": "out_for_delivery",                   // derived from status_code
  "incident_reason": null,                        // an OTEP incident reason (§4) when exception

  // WHO
  "actor": { "type": "driver", "name": "Jane D.", "phone": "+1..." },

  // PROVENANCE
  "source": {
    "type": "self_delivery",                     // self_delivery | third_party_delivery | carrier_label
    "provider_id": null,
    "carrier_code": null,
    "external_event_code": null,                 // your raw status code (preserve it)
    "raw": { /* original payload */ }
  },

  // PROOF
  "pod": {
    "photos": [ { "url": "...", "content_base64": null } ],
    "signature": [ { "url": "...", "content_base64": null } ],
    "recipient": "John Smith"
  }
}
```

Every field except `subject`, `occurred_at`, `status_code` and `source` is optional — partial sources fill what they have. Node/hub scans set `location` to the scanning facility. High-frequency GPS telemetry is NOT an OTEP event; an event is emitted on a status change or a node scan.

4. Vocabulary

4.1 Status codes

20 canonical lifecycle codes. `phase` is always derivable from the code.

code	phase	terminal	POD	meaning
`information_submitted`	pre_shipment			order info received
`booking_confirmed`	pre_shipment			carrier/booking acknowledged
`awaiting_pickup`	pre_shipment			ready for collection
`out_for_pickup`	pickup			en route to pick up
`picked_up`	pickup		✓	collected from shipper
`pickup_failed`	exception			pickup attempt failed
`pickup_rescheduled`	exception			pickup will retry
`received`	inbound			received at facility
`arrival_scan`	inbound			arrival scan at node
`in_transit`	transit			moving
`package_outbound`	transit			departed facility
`removed_from_route`	exception			taken off route
`route_cancelled`	exception			route cancelled
`out_for_delivery`	out_for_delivery			on the vehicle
`delivered`	delivered	✓	✓	delivered to consignee

code	phase	terminal	POD	meaning
`delivery_failed`	exception			delivery attempt failed
`delivery_rescheduled`	exception			will retry / redeliver
`return_to_sender`	return	✓		returning to origin
`rejected_by_recipient`	return	✓		recipient refused
`cancelled`	return	✓		order cancelled

4.2 Phases

`pre_shipment` · `preparing` · `pickup` · `inbound` · `in_custody` · `transit` · `out_for_delivery` · `delivered` · `exception` · `return`. (`preparing` and `in_custody` are reserved for non-parcel profiles — §8.)

4.3 Source types & time types

`source.type`: `self_delivery` · `third_party_delivery` · `carrier_label`. `time_type`: `actual` · `estimated` · `scheduled` (default `actual`).

4.4 Incident reasons

When an event is in the `exception` phase it SHOULD carry an `incident_reason` from this normalized vocabulary:

- Carrier: `carrier_damaged_parcel`, `carrier_sorting_error`, `carrier_address_not_found`, `carrier_parcel_lost`, `carrier_not_enough_time`, `carrier_vehicle_issue`, `carrier_capacity_exceeded`, `carrier_mechanical_delay`
- Retailer/shipper: `retailer_cancelled`, `retailer_incorrect_data`, `retailer_not_ready`, `retailer_incorrect_parcel`, `retailer_incorrect_dimensions`, `retailer_packaging_issue`
- Consignee: `consignee_refused`, `consignee_business_closed`, `consignee_not_available`, `consignee_not_home`, `consignee_cancelled`, `consignee_verification_failed`, `consignee_incorrect_address`, `consignee_access_restricted`, `consignee_safe_place_unavailable`
- Customs: `customs_delay`, `customs_documentation`, `customs_duties_unpaid`, `customs_prohibited`, `customs_inspection`
- Force majeure: `weather_delay`, `natural_disaster`, `force_majeure`
- Other: `parcel_being_researched`, `security_issue`, `regulatory_hold`, `unknown`

5. State machine & status-only sources

Phases advance forward; exceptions interrupt and resolve back. Terminal states (`delivered`, `return_to_sender`, `rejected_by_recipient`, `cancelled`) close the subject.

```
pre_shipment ? preparing ? pickup ? inbound ? in_custody ? transit ? out_for_delivery ? delivered ?
                                    ????????????? exception ??????????
                                    ????????????? return / cancelled ?
```

Rules:

- Consumers MUST order events by `occurred_at` and MUST tolerate out-of-order arrival.
- Transitions into a terminal state are idempotent; repeats are deduped.
- After a terminal status, a producer MUST NOT emit further events except a documented RMA / re-open flow.
- A source exposing only a current status MUST synthesize one event per observed change (with a stable dedupe key) rather than omit history. Over time the snapshots accumulate into a timeline.

6. External-standard mappings

OTEP's four dimensions line up field-for-field with the major standards, so each is an output projection of an OTEP event.

OTEP	GS1 EPCIS 2.0	IATA ONE Record	UN/CEFACT
`occurred_at` (+offset)	`eventTime` + `eventTimeZoneOffset`	`eventDate` + `eventTimeType=Actual`	Event Date/Time
`recorded_at`	`recordTime`	recordedAt	—
subject (sscc/piece)	`epcList` (SSCC URN)	`linkedObject` → Piece/Shipment	Consignment
`location` (gln/lat/lng)	`readPoint` / `bizLocation` (SGLN)	`recordedAtLocation` → Location	Location
`status_code`	`bizStep` + `disposition`	`eventCode`	Transport status code
`actor`	sourceList / extension	Actor / Party	Party
`incident_reason`	disposition / ErrorDeclaration	event remark	Status reason code

Per-code values (EPCIS CBV `bizStep` / `disposition`, ONE Record `eventCode`, UN/CEFACT code) are published in the machine-readable codebook. Beyond these international standards, an OTEP timeline can also be projected to OpenTelemetry traces, OGC SensorThings observations, and common commerce platforms (AfterShip, Shopify, Amazon, Walmart, BigCommerce, Magento, WooCommerce, Etsy).

> Confidence: EPCIS CBV values are stable standard URNs. ONE Record and UN/CEFACT code values are best-fit for last-mile and should be validated against the official code lists before external use. "Delivered to consignee" has no exact CBV bizStep — the closest fit (`receiving` + `received`) is used, or a user-vocabulary extension URN.

7. The OTEP API

OTEP is consumed over a small read-only HTTP surface; all endpoints are public.

Verb	Path	Returns
GET	`/api/v1/otep/trackings/{tracking_number}`	the timeline for a tracking number
GET	`/api/v1/otep/trackings/{tracking_number}/events`	events only
POST	`/api/v1/otep/trackings/batch`	many tracking numbers in one call
POST	`/api/v1/otep/validate`	conformance check for a posted timeline (§9)

A GraphQL query exposing the same timeline is also available.

Content negotiation

The same timeline is serialized into whichever representation you request, via a `?format=` query parameter or an `Accept` profile:

Request	Representation
`?format=otep` (default)	native OTEP timeline
`?format=epcis`	GS1 EPCIS 2.0 (JSON-LD `ObjectEvent` s)
`?format=onerecord`	IATA ONE Record (`LogisticsEvent` s)
`?format=uncefact`	UN/CEFACT transport status
`?format=otlp`	OpenTelemetry traces
`?format=sensorthings`	OGC SensorThings observations
`?format=aftership` | `shopify` | `amazon` | `walmart` | `bigcommerce` | `magento` | `woocommerce` | `etsy`	the platform's tracking/fulfillment shape

Events that cannot be assigned a code in a projection are skipped and counted (never silently dropped).

8. Domain profiles

OTEP is a general protocol, not a parcel one. The protocol layer (event envelope, phase spine, state machine) is universal; concrete status codes belong to a profile declared on the timeline via `profile`. The codes in §4 are the **`parcel`** profile. Other domains — moving, food delivery, storage and beyond — add their own code sets under their profile, namespaced `otep:<profile>:<code>`, each mapping up to the same phase spine. Adding a profile is an extension, not a protocol change.

9. Conformance specification (normative)

A producer or consumer is OTEP-conformant when every event it emits or accepts satisfies these tables and rules.

9.1 Timeline envelope

Field	Type	Req.	Constraints
`otep_version`	string	MUST	semver, e.g. `0.1`
`profile`	string	MUST	a registered profile
`subject`	object	MUST	§9.2
`current_status`	string|null	SHOULD	a status code (§4)
`current_phase`	string|null	SHOULD	MUST equal the phase of `current_status` if both present
`delivered`	boolean	SHOULD	`true` iff `current_status` = `delivered`
`events`	array	MUST	event objects (§9.3), orderable by `occurred_at`

9.2 `subject`

At least ONE of `tracking_number` / `order_id` / `package_id` MUST be present.

Field	Type	Constraints
`tracking_number`	string	non-empty
`order_id` / `package_id`	integer|null	
`external_tracking_number`	string|null	
`gs1_sscc`	string|null	18 digits
`piece_id`	string|null	

9.3 Event object

#	Field	Type	Req.	Constraints
1	`occurred_at`	string	MUST	ISO-8601 with offset
2	`recorded_at`	string|null	SHOULD	ISO-8601
3	`time_type`	string	MAY (default `actual`)	`actual` | `estimated` | `scheduled`
4	`status_code`	string|null	MUST[1]	a code in §4.1
5	`phase`	string|null	SHOULD	MUST equal the phase of `status_code`
6	`incident_reason`	string|null	SHOULD[2]	a reason in §4.4
7	`description`	string|null	MAY	human-readable
8	`location`	object|null	MAY	§9.4
9	`actor`	object|null	MAY	`{ type, name?, phone? }`; type ∈ {driver, operator, carrier, system}
10	`source`	object	MUST	§9.5
11	`pod`	object|null	MAY[3]	`{ photos[], signature[], recipient? }`

[1] A coded event MUST carry a status code from §4.1. A raw scan you cannot yet classify MAY set `status_code = null`, but MUST preserve the native code in `source.external_event_code` and MUST be counted, never dropped. [2] An `exception`-phase event SHOULD carry an `incident_reason`. [3] Events with `status_code` ∈ {`delivered`, `picked_up`} SHOULD carry a `pod`.

9.4 `location`

`name` (string) · `code` (string) · `gln` (GS1 GLN, 13 digits) · `lat` / `lng` (WGS-84) · `country` (ISO 3166-1 alpha-2). All optional.

9.5 `source`

Field	Type	Req.	Constraints
`type`	string	MUST	`self_delivery` | `third_party_delivery` | `carrier_label`
`provider_id`	integer | null	MAY	
`carrier_code`	string | null	MAY	
`external_event_code`	string | null	SHOULD[4]	your raw status code
`raw`	object | null	MAY	original payload

[4] MUST be present when `status_code` is null, so the native code is never lost.

9.6 Rules

1. Time. `occurred_at` MUST parse as ISO-8601. Normalize numeric epochs and `.NET /Date(ms)/` on ingest; do not emit those forms.
2. Ordering / dedup. Consumers MUST sort by `occurred_at`, tolerate out-of-order arrival, and dedupe on (`subject`, `status_code`, `occurred_at`).
3. State machine. After a terminal status, do not emit further events except a documented RMA / re-open.
4. No silent loss. Events that cannot be coded MUST be counted, never dropped.
5. Derivation. `phase`, `current_status`, `current_phase`, `delivered` are projections — if present they MUST be consistent with the event timeline.

9.7 Conformance levels & validation

- Level 1 — emits the envelope (§9.1), events with the required fields (§9.3), valid status codes (§4.1), valid phases (§4.2), and honors the state machine (§5).
- Level 2 — additionally emits at least one external-standard projection (§6) and, where applicable, profile-specific codes (§8).

Verify your output by POSTing a timeline to `POST /api/v1/otep/validate`. Treat any `errors` as blocking; address `warnings`. A machine-readable JSON Schema and the complete codebook (every status code, phase and external mapping) are published for offline validation.

10. Openness & governance

OTEP is an open specification, free for any party to implement.

- Normative vs informative. Normative: the event envelope, phase spine, status vocabulary, state machine and external-standard field mappings. How an implementer binds OTEP to its own internal systems is its own concern and out of scope here.
- Stable identifiers. Codes are addressed as `otep:<profile>:<code>`, phases as `otep:phase:<name>`. Once published in a released version, an identifier's meaning is immutable.
- Versioning. Semantic versioning. Adding codes/profiles is a MINOR (backward-compatible) change; changing the meaning of an existing code is a MAJOR change and SHOULD be avoided. The protocol version travels with every timeline (`otep_version`).

- Extension. New profiles and codes are proposed against this spec rather than forked, so independent implementers converge. Experimental codes MAY use an `x-` prefix (`otep:parcel:x-my_code`) until registered.
- License. The specification is intended to be released under an open license — TBD, pending sign-off.

11. Vendor interoperability — bring your own standard

OTEP welcomes other vendors to bring their own tracking-event standard so OTEP can interoperate with it, in either direction:

- Project OTEP → your standard. Define a mapping from an OTEP timeline to your format, reusing the OTEP status vocabulary. It is a pure transformation — timeline in, your structure out — so the mapping is written once and every OTEP producer can emit your format.
- Map your standard → OTEP. Provide a crosswalk from your status vocabulary onto the OTEP codes (§4) plus, if needed, a profile (§8). Your raw codes are preserved in `source.external_event_code`; unmapped codes are counted, never dropped.

Even if you cannot adopt OTEP directly, you are welcome to add a single normalized `otep_status` to your own API responses and to share your tracking event codes for crosswalk mapping. Propose mappings against this spec (rather than forking) so implementers converge; new formats plug into the same `?format=` content negotiation and never break an existing consumer.