OTEP — Open Tracking Event Protocol

便于阅读的译文 — 以英文版本为准。

> 状态：草案 v0.1 · 一份开放、厂商中立的规范 · 许可：开放（见 §10）

OTEP 是一个面向任何可追踪对象生命周期的开放、厂商中立的协议。
它定义了统一的事件模型和统一的状态词汇表，使得来自任何来源的追踪事件——自有车队配送、第三方快递、承运商面单及更多场景——都能够被交换、理

本文档即为该规范：包括结构、字段、状态表、状态机、外部标准映射，以及构建合规实现所需的一致性规则。它与具体实现无关——它描述的是协议本身，
2119 关键词（MUST / SHOULD / MAY）具有规范性。

1. OTEP 解决了什么

追踪是碎片化的：每家承运商对字段和状态码的命名各不相同，每个配送渠道以自己的形态上报，而对接全球标准则意味着一次又一次地重新集成。OTEP
为生产者和消费者提供了一种统一的共同语言：生产者只需发出一次 OTEP 事件，每个 OTEP
消费者都能理解它，并能将其投射到自己所需的标准。

2. 设计原则

1. 事件溯源（Event-sourced）。 事件时间线是事实的唯一来源；"当前状态"始终是一种投射——即最新事件的状态。
2. What / When / Where / Why。 每个事件都围绕这四个维度构建——它们与 GS1 EPCIS、IATA ONE Record 和 UN/CEFACT
   所共享的维度相同——因此这些标准是 OTEP 事件的输出投射，而非并行的模型。
3. 无列表来源不是障碍。 对于只暴露当前状态（没有历史）的来源，可通过为每次观测到的变化合成一个事件来处理（§5）。
4. 增量式（Additive）。 OTEP 与任何现有追踪 API 并存暴露；采用它绝不要求对消费者已经在使用的内容做破坏性变更。

3. OTEP 时间线

时间线是一个信封（envelope），承载被追踪的对象以及一组有序的事件。

```
{
  "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 */ ]
}
```

OTEP 事件

```
{
  // 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"
  }
}
```

除 `subject`、`occurred_at`、`status_code` 和 `source` 之外的每个字段都是可选的——部分来源只填充其拥有的内容。节点/枢纽扫描会将 `location` 设为执行扫描的设施。高频 GPS 遥测数据不是 OTEP 事件；事件是在状态发生变化或发生节点扫描时发出的。

4. 词汇表

4.1 状态码

20 个规范化的生命周期代码。`phase` 始终可由代码推导得出。

code	phase	终态	POD	含义
`information_submitted`	pre_shipment			已收到订单信息
`booking_confirmed`	pre_shipment			承运商/预订已确认
`awaiting_pickup`	pre_shipment			待揽收
`out_for_pickup`	pickup			正在前往揽收途中
`picked_up`	pickup		✓	已从发货方揽收
`pickup_failed`	exception			揽收尝试失败
`pickup_rescheduled`	exception			将重新尝试揽收
`received`	inbound			已在设施收货
`arrival_scan`	inbound			节点到达扫描
`in_transit`	transit			运输中
`package_outbound`	transit			已离开设施
`removed_from_route`	exception			已从路线中移除
`route_cancelled`	exception			路线已取消
`out_for_delivery`	out_for_delivery			已在配送车辆上
`delivered`	delivered	✓	✓	已送达收货人
`delivery_failed`	exception			配送尝试失败
`delivery_rescheduled`	exception			将重试/重新配送

code	phase	终态	POD	含义
`return_to_sender`	return	√		正在退回原点
`rejected_by_recipient`	return	√		收件人拒收
`cancelled`	return	√		订单已取消

4.2 阶段（Phases）

`pre_shipment` · `preparing` · `pickup` · `inbound` · `in_custody` · `transit` · `out_for_delivery` · `delivered` · `exception` · `return`。（`preparing` 和 `in_custody` 保留给非包裹类的 profile 使用——§8。）

4.3 来源类型与时间类型

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

4.4 异常原因（Incident reasons）

当事件处于 `exception` 阶段时，它 SHOULD 携带一个来自以下规范化词汇表的 `incident_reason`：

- 承运商：`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`
- 零售商/发货方：`retailer_cancelled`、`retailer_incorrect_data`、`retailer_not_ready`、`retailer_incorrect_parcel`、`retailer_incorrect_dimensions`、`retailer_packaging_issue`
- 收货人：`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`
- 海关：`customs_delay`、`customs_documentation`、`customs_duties_unpaid`、`customs_prohibited`、`customs_inspection`
- 不可抗力：`weather_delay`、`natural_disaster`、`force_majeure`
- 其他：`parcel_being_researched`、`security_issue`、`regulatory_hold`、`unknown`

5. 状态机与仅含状态的来源

阶段向前推进；异常会中断流程并解析后回退。终态（`delivered`、`return_to_sender`、`rejected_by_recipient`、`cancelled`）关闭该对象。

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

规则：

- 消费者 MUST 按 `occurred_at` 对事件排序，并且 MUST 容忍乱序到达。
- 进入终态的转换是幂等的；重复将被去重。
- 在终态之后，生产者 MUST NOT 发出更多事件，除非是有文档定义的 RMA / 重新开启流程。
- 只暴露当前状态的来源 MUST
  为每次观测到的变化合成一个事件（带有稳定的去重键），而不是省略历史。随着时间推移，这些快照会累积成一条时间线。

6. 外部标准映射

OTEP 的四个维度与主流标准逐字段对齐，因此每个标准都是 OTEP 事件的一种输出投射。

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

各代码对应的取值（EPCIS CBV `bizStep` / `disposition`、ONE Record `eventCode`、UN/CEFACT 代码）发布在机器可读的代码本（codebook）中。除了这些国际标准之外，一条 OTEP 时间线还可以投射到 OpenTelemetry 追踪、OGC SensorThings 观测，以及常见的电商平台（AfterShip、Shopify、Amazon、Walmart、BigCommerce、Magento、WooCommerce、Etsy）。

> 置信度：EPCIS CBV 取值是稳定的标准 URN。ONE Record 和 UN/CEFACT 的代码取值是针对最后一公里的最佳匹配，在对外使用前应对照官方代码列表进行校验。"送达收货人"没有完全精确的 CBV `bizStep` ——使用最接近的匹配（`receiving` + `received`），或使用用户词汇表扩展 URN。

7. OTEP API

OTEP 通过一个小型的只读 HTTP 接口集进行消费；所有端点均为公开的。

方法	路径	返回
GET	`/api/v1/otep/trackings/{tracking_number}`	某追踪号对应的时间线
GET	`/api/v1/otep/trackings/{tracking_number}/events`	仅返回事件
POST	`/api/v1/otep/trackings/batch`	一次调用查询多个追踪号
POST	`/api/v1/otep/validate`	对提交的时间线进行一致性校验（§9）

同时还提供一个暴露相同时间线的 GraphQL 查询。

内容协商（Content negotiation）

通过 `?format=` 查询参数或 `Accept` profile，同一条时间线会被序列化为你所请求的任意一种表示形式：

请求	表示形式
`?format=otep`（默认）	原生 OTEP 时间线
`?format=epcis`	GS1 EPCIS 2.0（JSON-LD `ObjectEvent`）
`?format=onerecord`	IATA ONE Record（`LogisticsEvent`）
`?format=uncefact`	UN/CEFACT 运输状态
`?format=otlp`	OpenTelemetry 追踪
`?format=sensorthings`	OGC SensorThings 观测
`?format=aftership` | `shopify` | `amazon` | `walmart` | `bigcommerce` | `magento` | `woocommerce` | `etsy`	对应平台的追踪/履约形态

在某个投射中无法被赋予代码的事件会被跳过并计数（绝不会被静默丢弃）。

8. 领域 Profile

OTEP 是一个通用协议，而非仅限于包裹的协议。协议层（事件信封、阶段主干、状态机）是通用的；而具体的状态码归属于在时间线上通过 `profile` 声明的某个 profile。§4 中的代码属于 **`parcel`** profile。其他领域——搬家、餐饮配送、仓储及更多——在各自的 profile 下添加自己的代码集，以 `otep:<profile>:<code>` 命名空间标识，每个代码都向上映射到同一套阶段主干。新增一个 profile 是一种扩展，而非协议变更。

9. 一致性规范（规范性）

当一个生产者或消费者发出或接受的每个事件都满足以下表格和规则时，它就是 OTEP-conformant（OTEP 一致的）。

9.1 时间线信封

字段	类型	要求	约束
`otep_version`	string	MUST	semver，例如 `0.1`
`profile`	string	MUST	已注册的 profile
`subject`	object	MUST	§9.2
`current_status`	string|null	SHOULD	一个状态码（§4）
`current_phase`	string|null	SHOULD	若两者都存在，则 MUST 等于 `current_status` 的 phase
`delivered`	boolean	SHOULD	当且仅当 `current_status = delivered` 时为 `true`
`events`	array	MUST	事件对象（§9.3），可按 `occurred_at` 排序

9.2 `subject`

`tracking_number` / `order_id` / `package_id` 中至少一个 MUST 存在。

字段	类型	约束
`tracking_number`	string	非空
`order_id` / `package_id`	integer|null	
`external_tracking_number`	string|null	
`gs1_sscc`	string|null	18 位数字
`piece_id`	string|null	

9.3 事件对象

#	字段	类型	要求	约束
1	`occurred_at`	string	MUST	带偏移量的 ISO-8601
2	`recorded_at`	string|null	SHOULD	ISO-8601
3	`time_type`	string	MAY（默认 `actual`）	`actual` | `estimated` | `scheduled`
4	`status_code`	string|null	MUST[1]	§4.1 中的一个代码
5	`phase`	string|null	SHOULD	MUST 等于 `status_code` 的 phase
6	`incident_reason`	string|null	SHOULD[2]	§4.4 中的一个原因
7	`description`	string|null	MAY	人类可读
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] 一个已编码的事件 MUST 携带一个来自 §4.1 的状态码。一次你尚无法分类的原始扫描 MAY 将 `status_code = null`，但 MUST 在 `source.external_event_code` 中保留原生代码，并 MUST 被计数，绝不丢弃。 [2] 处于 `exception` 阶段的事件 SHOULD 携带一个 `incident_reason`。 [3] `status_code` ∈ {`delivered`, `picked_up`} 的事件 SHOULD 携带一个 `pod`。

9.4 `location`

`name`（string）· `code`（string）· `gln`（GS1 GLN，13 位数字）· `lat` / `lng`（WGS-84）· `country`（ISO 3166-1 alpha-2）。全部可选。

9.5 `source`

字段	类型	要求	约束
`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]	你的原始状态码
`raw`	object|null	MAY	原始负载

[4] 当 `status_code` 为 null 时 MUST 存在，以确保原生代码不会丢失。

9.6 规则

1. 时间。 `occurred_at` MUST 能解析为 ISO-8601。在摄取时将数字纪元时间戳和 `.NET /Date(ms)/` 归一化；不要发出这些形式。
2. 排序 / 去重。 消费者 MUST 按 `occurred_at` 排序、容忍乱序到达，并基于（`subject`、`status_code`、`occurred_at`）去重。
3. 状态机。 在终态之后，不要发出更多事件，除非是有文档定义的 RMA / 重新开启。
4. 不静默丢失。 无法被编码的事件 MUST 被计数，绝不丢弃。
5. 派生。 `phase`、`current_status`、`current_phase`、`delivered` 都是投射——若存在，则它们 MUST 与事件时间线保持一致。

9.7 一致性级别与校验

- 级别 1 — 发出信封（§9.1）、带必需字段的事件（§9.3）、有效的状态码（§4.1）、有效的阶段（§4.2），并遵循状态机（§5）。
- 级别 2 — 此外还发出至少一种外部标准投射（§6），并在适用处发出特定于 profile 的代码（§8）。

校验你的输出：将一条时间线 POST 到 `POST /api/v1/otep/validate`。将任何 `errors` 视为阻断性问题；处理 `warnings`。还发布了机器可读的 JSON Schema 和完整的代码本（每个状态码、阶段和外部映射），用于离线校验。

10. 开放性与治理

OTEP 是一份开放规范，任何一方均可自由实现。

- 规范性 vs 信息性。 规范性：事件信封、阶段主干、状态词汇表、状态机以及外部标准字段映射。实现方如何将 OTEP 绑定到自身内部系统是其自己的事务，不在本文档范围之内。
- 稳定标识符。 代码以 `otep:<profile>:<code>` 寻址，阶段以 `otep:phase:<name>` 寻址。一旦在某个已发布版本中发布，标识符的含义即不可变。
- 版本管理。 采用语义化版本。新增代码/profile 是 MINOR（向后兼容）变更；改变现有代码的含义是 MAJOR 变更，且 SHOULD 避免。协议版本随每条时间线一起传递（`otep_version`）。
- 扩展。 新的 profile 和代码以针对本规范提案的方式提出，而非分叉，从而使各自独立的实现方收敛趋同。实验性代码 MAY 使用 `x-` 前缀（`otep:parcel:x-my_code`），直到正式注册为止。
- 许可。 本规范拟以开放许可发布——具体待定，等待签署确认。

11. 厂商互操作性 — 自带你的标准

OTEP 欢迎其他厂商带来各自的追踪事件标准，以便 OTEP 能与之互操作，两个方向皆可：

- 将 OTEP → 你的标准投射。 定义一个从 OTEP 时间线到你的格式的映射，复用 OTEP 状态词汇表。这是一个纯粹的转换——输入时间线，输出你的结构——因此该映射只需编写一次，每个 OTEP 生产者都能发出你的格式。
- 将你的标准 → OTEP 映射。 提供一个从你的状态词汇表到 OTEP 代码（§4）的对照表（crosswalk），并在需要时附带一个 profile（§8）。你的原始代码会保留在 `source.external_event_code` 中；未映射的代码会被计数，绝不丢弃。

即使你无法直接采用 OTEP，也欢迎你在自己的 API 响应中添加一个归一化的 `otep_status`，并分享你的追踪事件代码以用于对照映射。请以针对本规范提案的方式（而非分叉）来提出映射，使实现方收敛趋同；新格式会接入同一套 `?format=` 内容协商机制，且绝不会破坏任何现有的消费者。