# UNTP Lifecycle Events — GS1 EPCIS/CBV Implementation Mapping

# UNTP Lifecycle Events — GS1 EPCIS/CBV Implementation Mapping

## Purpose

The UNTP Digital Traceability Event (DTE) specification defines three lifecycle event types — **MakeEvent**, **MoveEvent**, and **ModifyEvent** — that can model any supply chain of any complexity. UNTP publishes its own JSON schema for these events, but recognises that many implementers have already invested in GS1 EPCIS 2.0 and the Core Business Vocabulary (CBV).

This document provides mapping guidance so that EPCIS/CBV implementers can express UNTP lifecycle events using their existing infrastructure without adopting the UNTP-native schema.

## Two Implementation Paths

| Approach | When to use |
|---|---|
| **UNTP-native schema** | Greenfield implementations, or where EPCIS is not already deployed. Use the `MakeEvent`, `MoveEvent`, and `ModifyEvent` JSON schemas published in `artefacts/schema/dte/`. |
| **GS1 EPCIS/CBV** | Organisations already operating an EPCIS 2.0 capture/query interface. Continue using EPCIS event types and CBV code lists, following the mapping rules below. |

Both paths produce credentials that are interoperable at the UNTP semantic level. The UNTP core vocabulary provides the linked-data bridge: EPCIS events can reference the same product, facility, and party identifiers and the same conformity credentials as UNTP-native events.

## Event Type Mapping

### MakeEvent → EPCIS TransformationEvent

A UNTP `MakeEvent` records the consumption of input products to create output products at a facility. The natural EPCIS equivalent is the `TransformationEvent`.

| UNTP property | EPCIS property | Notes |
|---|---|---|
| `type: MakeEvent` | `type: TransformationEvent` | |
| `id` | `eventID` | Unique event URI |
| `eventDate` | `eventTime` | ISO 8601 timestamp |
| `activityType` | `bizStep` | See bizStep mapping below |

| UNTP property | EPCIS property | Notes |
|---|---|---|
| `inputProduct[].product.id` | `inputEPCList[]` | Instance-level product identifiers (e.g. SGTIN) |
| `inputProduct[].quantity` | `inputQuantityList[]` | Class-level with `epcClass`, `quantity`, `uom` |
| `inputProduct[].disposition` | `disposition` (event-level) | See disposition mapping — UNTP tracks disposition per-product; EPCIS at event level |
| `outputProduct[].product.id` | `outputEPCList[]` | Instance-level product identifiers |
| `outputProduct[].quantity` | `outputQuantityList[]` | Class-level with `epcClass`, `quantity`, `uom` |
| `madeAtFacility.id` | `bizLocation` | GLN or facility URI |
| `relatedParty` | *(extension or master data)* | EPCIS does not have a native party-role field on events; use EPCIS master data or UNTP extension namespace |
| `relatedDocument` | `bizTransactionList` | Use `bizTransactionList` for linked business documents (DPPs, DCCs) |
| `sensorData` | `sensorElementList` | See sensor data mapping below |

### bizStep values for Make events

| Scenario | CBV `bizStep` | CBV URI |
|---|---|---|
| Manufacturing a new product (first use of identifier) | `commissioning` | `https://ref.gs1.org/cbv/BizStep-commissioning` |
| Creating additional quantity of an existing class | `creating_class_instance` | `https://ref.gs1.org/cbv/BizStep-creating_class_instance` |
| Assembling components (reversible) | `assembling` | `https://ref.gs1.org/cbv/BizStep-assembling` |

### Disposition values for Make events

Input products are implicitly consumed by the TransformationEvent. Set the event-level `disposition` to reflect the state of the **output** products. See Disposition Mapping for the full product-level vs event-level explanation.

| Scenario | CBV `disposition` | CBV URI |
|---|---|---|
| Output product newly commissioned | `active` | `https://ref.gs1.org/cbv/Disp-active` |

### Example: Smelter Make as EPCIS TransformationEvent

```
{
  "type": "TransformationEvent",
  "eventTime": "2025-03-05T06:00:00Z",
  "eventTimeZoneOffset": "+09:00",
  "bizStep": "https://ref.gs1.org/cbv/BizStep-commissioning",
  "disposition": "https://ref.gs1.org/cbv/Disp-active",
  "bizLocation": { "id": "https://facility-register.example.com/fac-002" },
  "inputQuantityList": [
    {
```

```
      "epcClass": "https://id.sample-mine.example.com/product/cu-conc-2025",
      "quantity": 30000,
      "uom": "KGM"
    }
  ],
  "outputQuantityList": [
    {
      "epcClass": "https://id.sample-refinery.example.com/product/cu-cathode-2025",
      "quantity": 10000,
      "uom": "KGM"
    }
  ],
  "bizTransactionList": [
    {
      "type": "https://ref.gs1.org/cbv/BTT-prodorder",
      "bizTransaction": "https://credentials.sample-refinery.example.com/dpp/cu-cathode-
        2025"
    }
  ]
}
```


---

## MoveEvent → EPCIS ObjectEvent (bizStep = shipping / receiving)

A UNTP `MoveEvent` records the shipment of products between two facilities. In EPCIS, a shipment is typically captured as an `ObjectEvent` with `bizStep = shipping` at the origin and optionally `bizStep = receiving` at the destination. The source and destination facilities are expressed via `sourceList` and `destinationList`.

| UNTP property | EPCIS property | Notes |
|---|---|---|
| `type: MoveEvent` | `type: ObjectEvent` | |
| `id` | `eventID` | Unique event URI |
| `eventDate` | `eventTime` | ISO 8601 timestamp |
| `activityType` | `bizStep` | See bizStep mapping below |
| `movedProduct[].product.id` | `epcList[]` | Instance-level product identifiers |
| `movedProduct[].quantity` | `quantityList[]` | Class-level with `epcClass`, `quantity`, `uom` |
| `movedProduct[].disposition` | `disposition` (event-level) | See disposition mapping — UNTP tracks disposition per-product; EPCIS at event level |
| `fromFacility.id` | `sourceList[].source` | With `type: https://ref.gs1.org/cbv/SDT-location` |
| `toFacility.id` | `destinationList[].destination` | With `type: https://ref.gs1.org/cbv/SDT-location` |
| `consignmentId` | `bizTransactionList[]` | Use `type: https://ref.gs1.org/cbv/BTT-bol` (Bill of Lading) |
| `relatedParty[role=shipper]` | `sourceList[]` | With `type: https://ref.gs1.org/cbv/SDT-possessing_party` |
| `relatedParty[role=receiver]` | `destinationList[]` | With `type: https://ref.gs1.org/cbv/SDT-` |

| UNTP property | EPCIS property | Notes |
|---|---|---|
| | | `possessing_party` |
| `relatedDocument` | `bizTransactionList` | Additional linked documents |
| `sensorData` | `sensorElementList` | See sensor data mapping below |

Note: EPCIS `ObjectEvent` requires an `action` field. Use `action: OBSERVE` for shipment events.

### bizStep values for Move events

| Scenario | CBV `bizStep` | CBV URI |
|---|---|---|
| General shipment (staging + loading + departing) | `shipping` | `https://ref.gs1.org/cbv/BizStep-shipping` |
| Shipment with change of ownership | `consigning` | `https://ref.gs1.org/cbv/BizStep-consigning` |
| Arrival at destination (before acceptance) | `arriving` | `https://ref.gs1.org/cbv/BizStep-arriving` |
| Receipt and acceptance into inventory | `receiving` | `https://ref.gs1.org/cbv/BizStep-receiving` |
| In-transit observation (e.g. sensor reading) | `transporting` | `https://ref.gs1.org/cbv/BizStep-transporting` |
| Loading into transport conveyance | `loading` | `https://ref.gs1.org/cbv/BizStep-loading` |
| Unloading from transport conveyance | `unloading` | `https://ref.gs1.org/cbv/BizStep-unloading` |
| Departure from origin | `departing` | `https://ref.gs1.org/cbv/BizStep-departing` |

### Disposition values for Move events

In UNTP, each moved product carries its own `disposition` (typically `in_transit` mapped to the `productStatus` code list). In EPCIS, set the event-level `disposition`. See Disposition Mapping for the full explanation.

| Scenario | CBV `disposition` | CBV URI |
|---|---|---|
| Products in transit | `in_transit` | `https://ref.gs1.org/cbv/Disp-in_transit` |
| Container sealed for shipment | `container_closed` | `https://ref.gs1.org/cbv/Disp-container_closed` |
| Container opened on arrival | `container_open` | `https://ref.gs1.org/cbv/Disp-container_open` |

### Example: Cathode Shipment as EPCIS ObjectEvent

```
{
  "type": "ObjectEvent",
  "action": "OBSERVE",
  "eventTime": "2025-03-10T14:00:00Z",
  "eventTimeZoneOffset": "+09:00",
  "bizStep": "https://ref.gs1.org/cbv/BizStep-shipping",
  "disposition": "https://ref.gs1.org/cbv/Disp-in_transit",
```

```
  "readPoint": { "id": "https://facility-register.example.com/fac-002" },
  "quantityList": [
    {
      "epcClass": "https://id.sample-refinery.example.com/product/cu-cathode-2025",
      "quantity": 2000,
      "uom": "KGM"
    }
  ],
  "sourceList": [
    {
      "type": "https://ref.gs1.org/cbv/SDT-location",
      "source": "https://facility-register.example.com/fac-002"
    },
    {
      "type": "https://ref.gs1.org/cbv/SDT-possessing_party",
      "source": "did:web:sample-refinery.example.com"
    }
  ],
  "destinationList": [
    {
      "type": "https://ref.gs1.org/cbv/SDT-location",
      "destination": "https://facility-register.example.com/fac-003"
    },
    {
      "type": "https://ref.gs1.org/cbv/SDT-possessing_party",
      "destination": "did:web:sample-battery.example.com"
    }
  ],
  "bizTransactionList": [
    {
      "type": "https://ref.gs1.org/cbv/BTT-bol",
      "bizTransaction": "urn:carrier:sea-freight:BL-2025-SG-BG-0310"
    },
    {
      "type": "https://ref.gs1.org/cbv/BTT-cert",
      "bizTransaction": "https://credentials.sample-refinery.example.com/dpp/cu-cathode-
        2025"
    }
  ]
}
```


---

## ModifyEvent → EPCIS ObjectEvent (various bizSteps)

A UNTP `ModifyEvent` records an action on a product that changes its state but retains its identity — such as inspection, repair, or testing. In EPCIS, this is also an `ObjectEvent`, but with a different `bizStep` and `disposition` to indicate the nature of the modification.

| UNTP property | EPCIS property | Notes |
|---|---|---|
| `type: ModifyEvent` | `type: ObjectEvent` | |
| `id` | `eventID` | Unique event URI |
| `eventDate` | `eventTime` | ISO 8601 timestamp |
| `activityType` | `bizStep` | See bizStep mapping below |
| `modifiedProduct[].product.id` | `epcList[]` | Instance-level product identifiers |
| `modifiedProduct[].quantity` | `quantityList[]` | Class-level with `epcClass`, `quantity`, `uom` |

| UNTP property | EPCIS property | Notes |
|---|---|---|
| `modifiedProduct[].disposition` | `disposition` (event-level) | See disposition mapping — UNTP tracks disposition per-product; EPCIS at event level |
| `modifiedAtFacility.id` | `bizLocation` | GLN or facility URI |
| `relatedDocument` | `bizTransactionList` | Linked test results, repair records, etc. |
| `sensorData` | `sensorElementList` | See sensor data mapping below |

Note: EPCIS `ObjectEvent` requires an `action` field. Use `action: OBSERVE` for inspections and tests. Use `action: ADD` for repairs that re-commission a product.

### bizStep values for Modify events

| Scenario | CBV `bizStep` | CBV URI |
|---|---|---|
| Non-destructive inspection or quality check | `inspecting` | `https://ref.gs1.org/cbv/BizStep-inspecting` |
| Repair of a malfunctioning product | `repairing` | `https://ref.gs1.org/cbv/BizStep-repairing` |
| Substitution of one product for another | `replacing` | `https://ref.gs1.org/cbv/BizStep-replacing` |
| Destructive quality testing or sampling | `sampling` | `https://ref.gs1.org/cbv/BizStep-sampling` |
| Disassembly into component parts | `disassembling` | `https://ref.gs1.org/cbv/BizStep-disassembling` |
| Removal from a composite object | `removing` | `https://ref.gs1.org/cbv/BizStep-removing` |
| Installation into a composite object | `installing` | `https://ref.gs1.org/cbv/BizStep-installing` |
| End of life / destruction | `destroying` | `https://ref.gs1.org/cbv/BizStep-destroying` |
| Decommissioning (removing identifier) | `decommissioning` | `https://ref.gs1.org/cbv/BizStep-decommissioning` |
| Collection for recycling or disposal | `collecting` | `https://ref.gs1.org/cbv/BizStep-collecting` |
| Segregation pending review | `holding` | `https://ref.gs1.org/cbv/BizStep-holding` |
| Dispensing product to consumer | `dispensing` | `https://ref.gs1.org/cbv/BizStep-dispensing` |
| Repackaging into different configuration | `repackaging` | `https://ref.gs1.org/cbv/BizStep-repackaging` |
| IoT sensor reporting (standalone) | `sensor_reporting` | `https://ref.gs1.org/cbv/BizStep-sensor_reporting` |

### Disposition values for Modify events

In UNTP, each modified product carries its own `disposition` (from the `productStatus` code list: `repaired`, `disposed`, etc.). In EPCIS, set the event-level `disposition`. If a single UNTP ModifyEvent has products with different dispositions, split into separate EPCIS ObjectEvents. See Disposition Mapping for the full explanation.

| Scenario | CBV `disposition` | CBV URI |
|---|---|---|
| Passed inspection / conformant | `conformant` | `https://ref.gs1.org/cbv/Disp-conformant` |
| Failed inspection | `non_conformant` | `https://ref.gs1.org/cbv/Disp-non_conformant` |
| Returned to service after repair | `available` | `https://ref.gs1.org/cbv/Disp-available` |
| Removed from service pending repair | `unavailable` | `https://ref.gs1.org/cbv/Disp-unavailable` |
| Needs replacement | `needs_replacement` | `https://ref.gs1.org/cbv/Disp-needs_replacement` |
| Physical damage detected | `damaged` | `https://ref.gs1.org/cbv/Disp-damaged` |
| Product destroyed | `destroyed` | `https://ref.gs1.org/cbv/Disp-destroyed` |
| Product expired | `expired` | `https://ref.gs1.org/cbv/Disp-expired` |
| Recalled for safety reasons | `recalled` | `https://ref.gs1.org/cbv/Disp-recalled` |
| Decommissioned (may be re-commissioned) | `inactive` | `https://ref.gs1.org/cbv/Disp-inactive` |
| Dispensed to consumer (full quantity) | `dispensed` | `https://ref.gs1.org/cbv/Disp-dispensed` |
| Partially dispensed | `partially_dispensed` | `https://ref.gs1.org/cbv/Disp-partially_dispensed` |
| Returned by customer | `returned` | `https://ref.gs1.org/cbv/Disp-returned` |
| Disposed for destruction | `disposed` | `https://ref.gs1.org/cbv/Disp-disposed` |

### Example: Battery Inspection as EPCIS ObjectEvent


```
{
  "type": "ObjectEvent",
  "action": "OBSERVE",
  "eventTime": "2025-04-02T10:00:00Z",
  "eventTimeZoneOffset": "+02:00",
  "bizStep": "https://ref.gs1.org/cbv/BizStep-inspecting",
  "disposition": "https://ref.gs1.org/cbv/Disp-conformant",
  "bizLocation": { "id": "https://facility-register.example.com/fac-003" },
  "epcList": [
    "https://id.sample-battery.example.com/product/bat-75kwh-2025/item/BAT-75-2025-00471"
  ],
  "bizTransactionList": [
    {
      "type": "https://ref.gs1.org/cbv/BTT-testres",
      "bizTransaction": "https://credentials.sample-battery.example.com/dcc/bat-inspection-00471"
    }
  ]
}
```

# Disposition Mapping — Product-level vs Event-level

A key structural difference between UNTP and EPCIS is where product disposition (the state of the product after the event) is recorded:

- **UNTP** tracks disposition **per product** via the `disposition` property on `EventProduct`, using the `productStatus` code list. This means a single event can record different disposition values for different products — for example, a MakeEvent can mark input products as `consumed` and output products as `new` in the same event.
- **EPCIS** tracks disposition **per event** via the `disposition` property on the event itself, using the CBV `Disp` vocabulary. All products in the event share the same disposition. When different products need different dispositions, separate EPCIS events must be created.

## UNTP productStatus → CBV Disposition

The UNTP `productStatus` code list has five values. The table below maps each to the most appropriate CBV disposition(s).

| UNTP `productStatus` | Meaning | CBV `disposition` | CBV URI | Typical event context |
|---|---|---|---|---|
| `new` | Newly created product | `active` | `https://ref.gs1.org/cbv/Disp-active` | Output of a MakeEvent (TransformationEvent) |
| `consumed` | Input material fully consumed | *(no direct CBV equivalent)* | — | Input of a MakeEvent. In EPCIS the input products are implicitly consumed by the TransformationEvent. No separate disposition is needed. |
| `repaired` | Product returned to service after repair | `available` | `https://ref.gs1.org/cbv/Disp-available` | ModifyEvent with `bizStep = repairing` |
| `recycled` | Product recycled into new material | `active` | `https://ref.gs1.org/cbv/Disp-active` | Output of a MakeEvent where inputs are recycled material. Alternatively use `disposed` for the recycled input. |
| `disposed` | Product removed from service | `disposed` | `https://ref.gs1.org/cbv/Disp-disposed` | ModifyEvent with `bizStep = destroying` or `collecting`. Also consider `destroyed` if physically destroyed. |

## Handling the structural difference

When mapping a single UNTP event that contains products with **different dispositions** to EPCIS:

1. **MakeEvent** — the EPCIS `TransformationEvent` inherently handles this. Input products are implicitly consumed (they disappear from the supply chain), and the event-level `disposition` (e.g. `active`) applies to the output products. No separate events are needed.

2. **MoveEvent** — typically all moved products share the same disposition (`in_transit`), so the event-level disposition works naturally.

3. **ModifyEvent** — if a single UNTP ModifyEvent lists products with different dispositions (e.g. one product `repaired`, another `disposed`), split into separate EPCIS ObjectEvents, each with the appropriate event-level `disposition`.

### CBV dispositions relevant to UNTP use cases

Beyond the five UNTP `productStatus` values, EPCIS implementers may use additional CBV dispositions that provide finer granularity:

| CBV `disposition` | CBV URI | Relevant UNTP scenario |
|---|---|---|
| `in_transit` | `https://ref.gs1.org/cbv/Disp-in_transit` | MoveEvent — products being shipped |
| `conformant` | `https://ref.gs1.org/cbv/Disp-conformant` | ModifyEvent — passed quality inspection |
| `non_conformant` | `https://ref.gs1.org/cbv/Disp-non_conformant` | ModifyEvent — failed quality inspection |
| `damaged` | `https://ref.gs1.org/cbv/Disp-damaged` | ModifyEvent — physical damage detected |
| `needs_replacement` | `https://ref.gs1.org/cbv/Disp-needs_replacement` | ModifyEvent — component must be replaced |
| `unavailable` | `https://ref.gs1.org/cbv/Disp-unavailable` | ModifyEvent — removed from service pending repair |
| `destroyed` | `https://ref.gs1.org/cbv/Disp-destroyed` | ModifyEvent — physically destroyed |
| `recalled` | `https://ref.gs1.org/cbv/Disp-recalled` | ModifyEvent — recalled for safety |
| `expired` | `https://ref.gs1.org/cbv/Disp-expired` | ModifyEvent — past expiration date |
| `returned` | `https://ref.gs1.org/cbv/Disp-returned` | ModifyEvent — returned by customer |
| `inactive` | `https://ref.gs1.org/cbv/Disp-inactive` | ModifyEvent — decommissioned |
| `container_closed` | `https://ref.gs1.org/cbv/Disp-container_closed` | MoveEvent — container sealed for shipment |

These finer CBV values can be used by EPCIS implementers to provide richer disposition semantics than the five UNTP `productStatus` codes. When converting back to UNTP, map to the closest `productStatus` value.

---

## Sensor Data Mapping

Both UNTP and EPCIS support IoT sensor data on events. The structures are conceptually aligned but differ in shape.

| UNTP `sensorData` property | EPCIS `sensorElementList` property | Notes |
|---|---|---|
| `metric.id` | `sensorReport[].type` (measurementType) | EPCIS uses `gs1:MeasurementType` URIs |
| `metric.name` | *(label from vocabulary)* | EPCIS relies on the measurement type vocabulary |
| `measure[].value` | `sensorReport[].value` | Numeric value |
| `measure[].unit` | `sensorReport[].uom` | UN/CEFACT Rec 20 code |
| `sensor.id` | `sensorMetadata.deviceID` | Sensor device URI |
| `geoLocation.latitude` | `sensorReport[].component: latitude` | EPCIS uses component decomposition for coordinates |
| `geoLocation.longitude` | `sensorReport[].component: longitude` | |
| `rawData[].linkURL` | `sensorMetadata.rawData` | Link to raw sensor data file |

### Example: Temperature Sensor as EPCIS SensorElement

```
{
  "sensorElementList": [
    {
      "sensorMetadata": {
        "deviceID": "https://sensors.sample-refinery.example.com/temp-401",
        "rawData": "https://sensors.sample-refinery.example.com/temp-401/raw/2025-03-05"
      },
      "sensorReport": [
        {
          "type": "gs1:Temperature",
          "value": 22.5,
          "uom": "CEL"
        },
        {
          "type": "gs1:Temperature",
          "value": 23.1,
          "uom": "CEL"
        }
      ]
    }
  ]
}
```

---

# Product Identification Mapping

UNTP `EventProduct` wraps a `Product` with quantity and disposition. EPCIS separates instance-level identifiers (`epcList`) from class-level quantities (`quantityList`).

| UNTP `EventProduct` | EPCIS equivalent | When to use |
|---|---|---|
| `product.id` + `idGranularity: item` | `epcList[]` entry (e.g. SGTIN URI) | Serialised individual items |
| `product.id` + `idGranularity: model` + `quantity` | `quantityList[]` entry with `epcClass` (e.g. GTIN URI), `quantity`, `uom` | Batch or class-level quantities |

| UNTP `EventProduct` | EPCIS equivalent | When to use |
|---|---|---|
| `product.batchNumber` | `epcClass` using LGTIN pattern | Batch-level identification |
| `disposition` | Event-level `disposition` | EPCIS disposition applies to all products in the event, not per-product |

### GS1 Identifier Patterns

| UNTP identifier | GS1 EPC URI pattern | Example |
|---|---|---|
| Serialised item | `urn:epc:id:sgtin:{companyPrefix}.{itemRef}.{serial}` | `urn:epc:id:sgtin:0614141.107346.2025001` |
| Product class (GTIN) | `urn:epc:idpat:sgtin:{companyPrefix}.{itemRef}.*` | `urn:epc:idpat:sgtin:0614141.107346.*` |
| Batch/Lot (LGTIN) | `urn:epc:class:lgtin:{companyPrefix}.{itemRef}.{lot}` | `urn:epc:class:lgtin:0614141.107346.2025-Q1` |
| Facility (GLN) | `urn:epc:id:sgln:{companyPrefix}.{locationRef}.{extension}` | `urn:epc:id:sgln:0614141.12345.0` |

---

# Business Transaction Type Mapping

EPCIS `bizTransactionList` can reference UNTP credentials by treating them as linked business documents. Use these CBV business transaction types:

| UNTP document type | CBV `bizTransactionType` | CBV URI |
|---|---|---|
| Digital Product Passport (DPP) | `cert` (Certificate) | `https://ref.gs1.org/cbv/BTT-cert` |
| Digital Conformity Credential (DCC) | `cert` (Certificate) | `https://ref.gs1.org/cbv/BTT-cert` |
| Bill of Lading / consignment | `bol` (Bill of Lading) | `https://ref.gs1.org/cbv/BTT-bol` |
| Purchase order | `po` (Purchase Order) | `https://ref.gs1.org/cbv/BTT-po` |
| Despatch advice / ASN | `desadv` (Despatch Advice) | `https://ref.gs1.org/cbv/BTT-desadv` |
| Production order | `prodorder` (Production Order) | `https://ref.gs1.org/cbv/BTT-prodorder` |
| Test procedure | `testprd` (Test Procedure) | `https://ref.gs1.org/cbv/BTT-testprd` |
| Test result | `testres` (Test Result) | `https://ref.gs1.org/cbv/BTT-testres` |
| Upstream EPCIS event | `upevt` (Upstream EPCIS Event) | `https://ref.gs1.org/cbv/BTT-upevt` |

---

# Facility and Party Mapping

| UNTP property | EPCIS property | Notes |
|---|---|---|
| `madeAtFacility` / `modifiedAtFacility` | `bizLocation` | The facility where the event occurred. Use GLN or facility register URI. |
| `fromFacility` | `readPoint` + `sourceList[type=location]` | Origin of a shipment |
| `toFacility` | `destinationList[type=location]` | Destination of a shipment |
| `relatedParty[role=shipper]` | `sourceList[type=possessing_party]` | Party who shipped the goods |
| `relatedParty[role=receiver]` | `destinationList[type=possessing_party]` | Party who received the goods |
| `relatedParty[role=manufacturer]` | *(master data or extension)* | EPCIS does not have a native manufacturer role on events; attach via ILMD or master data |

---

# Wrapping EPCIS Events as Verifiable Credentials

UNTP requires all credentials to be issued as W3C Verifiable Credentials. EPCIS implementers have two options:

1. **Wrap the EPCISDocument in a VC envelope.** The entire EPCIS JSON-LD document becomes the `credentialSubject` of a `VerifiableCredential`. This preserves the EPCIS structure verbatim while adding the VC trust layer (issuer, proof, validity).

2. **Use EPCIS alongside UNTP-native DTEs.** Issue UNTP-native DTE credentials for the UNTP trust layer and link them to existing EPCIS repositories via `relatedDocument` links. This approach works well when the EPCIS repository is already established and the VC layer is added on top.

```
{
  "type": ["VerifiableCredential"],
  "@context": [
    "https://www.w3.org/ns/credentials/v2",
    "https://ref.gs1.org/epcis/"
  ],
  "id": "https://credentials.sample-refinery.example.com/dte/make-cathode-2025-0305",
  "issuer": {
    "id": "did:web:sample-refinery.example.com",
    "name": "Sample Copper Refinery Co. Ltd"
  },
  "validFrom": "2025-03-05T00:00:00Z",
  "credentialSubject": {
    "type": "TransformationEvent",
    "...": "... (EPCIS event as above)"
  }
}
```

---

# Summary Mapping Table

| UNTP Event Type | EPCIS Event Type | EPCIS `action` | Primary `bizStep` values | Primary `disposition` values |
|---|---|---|---|---|
| **MakeEvent** | `TransformationEvent` | *(none)* | `commissioning`, `creating_class_instance`, `assembling` | `active` |
| **MoveEvent** | `ObjectEvent` | `OBSERVE` | `shipping`, `receiving`, `departing`, `arriving`, `transporting` | `in_transit`, `container_closed` |
| **ModifyEvent** | `ObjectEvent` | `OBSERVE` or `ADD` | `inspecting`, `repairing`, `sampling`, `destroying`, `decommissioning` | `conformant`, `non_conformant`, `available`, `damaged`, `destroyed` |

---

# References

- EPCIS 2.0 Ontology (JSON-LD) — GS1 EPCIS linked data vocabulary
- CBV 2.0 Ontology (JSON-LD) — GS1 Core Business Vocabulary (bizStep, disposition, etc.)
- EPCIS 2.0 Standard — Full specification document
- UNTP DTE Schema — UNTP-native JSON schemas
- UNTP Core Vocabulary — Shared classes and properties