# Event Schemas & Message Contracts

## Purpose & Scope

This document defines the asynchronous event schemas and message contracts used by the PineCone Pro Supplies ERP/IMS. The system relies on a **publish‑subscribe** event bus to propagate domain changes between loosely coupled microservices. Each microservice publishes domain events when it updates its own state and subscribes to events that it needs to react to. A clear event schema helps guarantee **eventual consistency** and traceability while keeping the services autonomous. The scope of this document includes:

* The core principles and standards used to design the events.
* Naming conventions, metadata and payload guidelines.
* A generic event envelope with metadata and data sections.
* Domain‑specific event definitions and message contracts for all MVP features and Phase 2 enhancements (e.g., product information management, inventory, orders, purchasing, lot/serial tracking, shipping, returns, tax/accounting, analytics and future forecasting/promotions/kitting/3PL/customer service/EDI).

The document complements the API specifications by describing the asynchronous communication patterns between services. It is not tied to a specific messaging technology; events may be transported via Azure Service Bus, RabbitMQ, or other brokers as long as the envelope format is respected【684820280994233†L338-L349】.

## Architectural Context

The ERP/IMS follows an **event‑driven microservices** architecture. When a microservice performs a notable business operation (e.g., creating an order or receiving inventory), it publishes an event describing what happened. Other services subscribe to those events and update their own state accordingly, enabling **eventual consistency** across the system【684820280994233†L338-L356】. An event bus provides the publish/subscribe mechanism, decoupling producers from consumers【684820280994233†L343-L349】. Transactions that span multiple services are decomposed into a series of distributed actions; each action emits an event to trigger the next step【684820280994233†L351-L356】. Because the underlying data stores are isolated, services must handle duplicate events (idempotence) and ensure that they can recover gracefully from network errors or retries【684820280994233†L351-L356】.

## Event Design Guidelines & Conventions

### Domain events vs. commands

Domain events describe something **that already happened** in the business domain. They are named in the past tense and should not encode instructions about what should happen next; that responsibility belongs to commands or queries. For example, order.created is an event, whereas create‑order would be a command. Events should emerge naturally from **process walkthroughs** or **event‑storming sessions**, and they should reflect the ubiquitous language of each bounded context【806325492643920†L100-L115】.

### Naming conventions

To ensure clarity and consistency, the ERP/IMS adopts naming rules inspired by the CloudEvents specification and domain‑driven design:

* **Namespace:** prefix each event type with a reverse‑DNS domain representing the organization and service context (e.g., com.pinecone.pim.product-created). This makes event types globally unique and easier to route【458455304652570†L137-L147】.
* **Lowercase & hyphens:** use lowercase letters and hyphens to separate words; avoid camelCase, underscores or dots in the event name【458455304652570†L137-L151】.
* **Past‑tense verb:** include a verb in past tense to indicate that the event represents something that has already happened (e.g., product-created, order-paid)【458455304652570†L153-L159】. Do not start the event name with the verb; embed the verb naturally within the domain concept (e.g., user-added-to-group rather than add-user-to-group)【458455304652570†L161-L168】.
* **Avoid generic verbs:** avoid vague names such as created or updated; choose expressive domain language (e.g., price-changed instead of product-updated)【458455304652570†L169-L171】.
* **Versioning:** encode the event version in the type when a breaking change is made (e.g., .v1, .v2). Multiple versions may coexist during migration【458455304652570†L174-L186】.

### Event envelope & metadata

Following the pattern recommended by Amazon EventBridge and other event brokers, events are divided into two sections: **metadata** and **data**【806325492643920†L152-L159】. The metadata contains information used for routing, filtering and observability, while the data contains the domain‑specific payload. The use of metadata helps with **filtering**, **downstream processing** and **debugging**【806325492643920†L163-L169】. Key metadata fields include:

| Field | Description & guidance | Source |
| --- | --- | --- |
| eventId | Unique identifier for the event. Producers must ensure that the combination of source and eventId is unique for each event【280384752129849†L303-L315】. Typically a UUID or event counter is used. | CloudEvents spec【280384752129849†L303-L319】 |
| eventType | Name of the event following the naming conventions above. It describes the type of occurrence and is used for routing and subscription filtering【280384752129849†L380-L399】. | CloudEvents spec【280384752129849†L380-L399】 |
| eventVersion | Semantic version of the event schema (e.g., 1.0). When breaking changes occur, publish both old and new versions concurrently to allow consumers to migrate【806325492643920†L257-L318】. | AWS Event Design【806325492643920†L257-L318】 |
| source | URI or reverse‑DNS string identifying the origin of the event (service/bounded context). Combined with eventId it uniquely identifies the event【280384752129849†L320-L343】. | CloudEvents spec【280384752129849†L320-L343】 |
| domain / service | Names of the business domain and service raising the event. Including both improves observability and helps filter events by domain or service【806325492643920†L198-L213】. | AWS Event Design【806325492643920†L198-L213】 |
| timestamp | UTC timestamp when the event occurred. All services must use consistent algorithms to set this field【280384752129849†L505-L518】. | CloudEvents spec【280384752129849†L505-L518】 |
| correlationId | Identifier that ties together multiple events related to the same business process (e.g., all events related to processing a single order). It can be provided externally or generated by the initial request. Correlation IDs enable tracing across services【806325492643920†L214-L224】. | AWS Event Design【806325492643920†L214-L224】 |
| requestId | Unique identifier for this request or transaction, generated internally for each inbound request. Useful for tracing retries and sagas【806325492643920†L214-L224】. | AWS Event Design【806325492643920†L214-L224】 |
| metadata.version | Repeated from eventVersion (explicitly included in metadata for filtering). | AWS Event Design【806325492643920†L257-L318】 |
| additionalMetadata | A flexible map for extra attributes such as user identity, tenant ID or security context. Extension attributes follow the CloudEvents naming and type rules【280384752129849†L522-L555】. | CloudEvents spec【280384752129849†L522-L555】 |

#### Example envelope

{  
 "metadata": {  
 "eventId": "4f81c9b0-9e1d-11ed-93f3-5bd98734f1ab",  
 "eventType": "com.pinecone.orders.order-created",  
 "eventVersion": "1.0",  
 "source": "com.pinecone.orders",  
 "domain": "Orders",  
 "service": "OrderService",  
 "timestamp": "2025-09-27T14:32:15Z",  
 "correlationId": "ccd5bf2c-5e32-4f11-8a22-0a1cbbf2c7e4",  
 "requestId": "5ed0b9e0-9e1d-11ed-93f3-5bd98734f1ab",  
 "additionalMetadata": {  
 "userId": "U123456",  
 "tenantId": "T001"  
 }  
 },  
 "data": {  
 "orderId": 100123,  
 "customerId": "C1000",  
 "orderDate": "2025-09-27",  
 "status": "Pending",  
 "items": [  
 {"productId": 20045, "quantity": 2, "unitPrice": 10.99},  
 {"productId": 20067, "quantity": 1, "unitPrice": 25.00}  
 ]  
 }  
}

### Event versioning & evolution

Events are immutable records; they cannot be changed once published. When event schemas need to evolve, producers MUST follow a **versioned event** approach by incrementing the eventVersion and publishing both the old and new versions during the migration period【806325492643920†L257-L318】. Consumers must be tolerant of additional fields (backwards‑compatible changes) and filter on eventVersion if they only support specific versions. Avoid making breaking changes unnecessarily; prefer adding optional fields or using separate events for new concepts.

### Handling large or sensitive payloads

Large payloads and personally identifiable information should not be included directly in the event. If the payload exceeds broker limits or contains sensitive data, the producer uploads the full payload to secure storage (e.g., Azure Blob Storage or S3) and includes a **pre‑signed URL** in the event. This “claim check” pattern allows services to fetch the data on demand while keeping the event message small and auditable【806325492643920†L323-L372】.

### Idempotence & duplicate handling

Consumers MUST handle duplicate events gracefully. Since network retries may deliver the same event more than once, consumers should use the combination of eventId and source to detect duplicates【280384752129849†L303-L315】. Side effects (e.g., database inserts) must be idempotent.

### Autonomy & library sharing

As Microsoft’s guidance warns, integration events should be defined within each microservice rather than in a shared library, to maintain autonomy and avoid coupling【684820280994233†L437-L443】. Only infrastructure libraries such as the event bus client and JSON serializers are shared【684820280994233†L451-L455】.

## Event Schema Definition

Each event in the system follows the envelope pattern described above. The **metadata** section conforms to the CloudEvents context attributes specification and includes additional fields for correlation and versioning. The **data** section contains domain‑specific information. Below is a summary of mandatory and optional fields:

| Section | Field | Type | Required | Description |
| --- | --- | --- | --- | --- |
| Metadata | eventId | string (UUID) | Yes | Unique identifier for the event【280384752129849†L303-L315】. |
| Metadata | eventType | string | Yes | Fully qualified event type (reverse‑DNS namespace + domain‑specific name)【280384752129849†L380-L399】【458455304652570†L137-L159】. |
| Metadata | eventVersion | string | Yes | Semantic version of the event schema【806325492643920†L257-L318】. |
| Metadata | source | string (URI) | Yes | Identifier of the service or context where the event occurred【280384752129849†L320-L343】. |
| Metadata | domain | string | Yes | Business domain (e.g., “Orders”, “Inventory”). |
| Metadata | service | string | Yes | Name of the microservice raising the event【806325492643920†L198-L213】. |
| Metadata | timestamp | RFC 3339 timestamp | Yes | Time of the occurrence【280384752129849†L505-L518】. |
| Metadata | correlationId | string | Optional | Traces related events across services【806325492643920†L214-L224】. |
| Metadata | requestId | string | Optional | Unique identifier for the current request【806325492643920†L214-L224】. |
| Metadata | additionalMetadata | object | Optional | Map of additional attributes (user, tenant, etc.)【280384752129849†L522-L555】. |
| Data | — | object | Yes | Domain‑specific payload; fields defined per event. |

## Domain‑Specific Events & Message Contracts

The following sections list the events for each major module. Each event definition includes a brief description, the producer (service that raises the event), typical consumers, and the payload schema. Field names use camelCase within the payload for readability. Optional fields are marked with “(opt)”.

### 1. Product Information Management (PIM)

| Event Type | Description & Trigger | Producer | Consumers | Payload Fields |
| --- | --- | --- | --- | --- |
| **com.pinecone.pim.product-created** | Emitted when a new product/SKU (including kits or bundles) is created in the PIM. | PIM service | Inventory, Order, Pricing, Search index | productId (int), sku (string), name (string), categoryId (int), unitOfMeasure (string), hazmatClass (string opt), listPrice (decimal), bundleComponents (array of {componentSku: string, quantity: decimal} opt), createdBy (string) |
| **com.pinecone.pim.product-updated** | Indicates that one or more product attributes have changed (name, description, category, UOM, hazmat class). Use more specific events like price‑changed when appropriate. | PIM service | Inventory, Order, Pricing | productId, updatedFields (array of strings), newValues (object), oldValues (object) |
| **com.pinecone.pim.product-price-changed** | Raised when a product’s list price or cost price is updated. | PIM service | Pricing, Order, Promotions | productId, oldPrice, newPrice, effectiveDate |
| **com.pinecone.pim.product-deleted** | Product/SKU has been discontinued. Consumers should archive or disable references. | PIM service | Inventory, Order | productId, deletedAt |
| **com.pinecone.pim.product-hazmat-updated** | Hazardous classification or handling instructions changed. | PIM service | Shipping, Compliance, Warehouse | productId, oldHazmatClass, newHazmatClass, effectiveDate |

### 2. Inventory & Warehouse Management

| Event Type | Description & Trigger | Producer | Consumers | Payload Fields |
| --- | --- | --- | --- | --- |
| **com.pinecone.inventory.item-received** | New stock arrives at a warehouse (purchase order receipt, cross‑dock or returns). | Inventory service | Purchasing, Order service, Analytics | inventoryItemId (int), productId, receivedQty (decimal), unitOfMeasure, warehouseId, lotNumber (string opt), serialNumbers (array opt), receivedDate, poId (opt), putawayLocation (string opt) |
| **com.pinecone.inventory.item-adjusted** | Adjustment to on‑hand quantity due to cycle count, shrinkage, damage or manual correction. | Inventory service | Accounting, Analytics | inventoryItemId, productId, adjustmentQty (decimal; positive for gain, negative for loss), reasonCode (string), warehouseId, performedBy |
| **com.pinecone.inventory.transfer-initiated** | Initiates a stock transfer between locations/warehouses (includes cross‑dock shipments). | Inventory service | Warehouse, Shipping | transferId, sourceWarehouseId, destinationWarehouseId, lineItems (array of {productId, quantity}), initiatedBy |
| **com.pinecone.inventory.transfer-completed** | Confirms that a transfer has arrived and stock is available at the destination. | Inventory service | Order service, Analytics | transferId, receivedDate, confirmedBy |
| **com.pinecone.inventory.cyclecount-completed** | Cycle count results for a location or warehouse. Used to update inventory accuracy metrics. | Inventory service | Analytics | countId, warehouseId, locationId (opt), countDate, items (array of {productId, countedQty, expectedQty}), performedBy |
| **com.pinecone.inventory.lot-expired** | A lot has reached its expiration date; stock must be quarantined or removed. | Inventory service | Purchasing, Order, Compliance | lotNumber, productId, expiredQty, warehouseId, expirationDate |
| **com.pinecone.inventory.serial-assigned** | A serialised item has been assigned to a specific product instance (e.g., a tool). | Inventory service | Order service, Warranty | serialNumber (string), productId, assignedToOrderId (opt), assignedDate |

### 3. Order Management & Fulfillment

| Event Type | Description & Trigger | Producer | Consumers | Payload Fields |
| --- | --- | --- | --- | --- |
| **com.pinecone.orders.order-created** | A new B2B/B2C order has been placed via the storefront, POS, portal or API. | Order service | Payment, Inventory, CRM, Fulfillment | orderId (int), customerId (string), orderDate, channel (e.g., web, portal, POS, Amazon), totalAmount, currency, items (array of {productId, quantity, unitPrice}), shippingAddress, billingAddress, paymentMethod (opt) |
| **com.pinecone.orders.order-status-updated** | The order status has changed (e.g., Pending → Paid, Picking → Shipped). Multiple events may be published for each transition. | Order service | Customer service, Warehouse, Analytics | orderId, oldStatus, newStatus, statusChangedAt, reason (opt) |
| **com.pinecone.orders.order-paid** | Payment has been authorised and captured for the order. Includes fraud screen result. | Payment service | Order service, Accounting | orderId, paymentId, paymentMethod, amountPaid, currency, paymentDate, fraudCheckStatus |
| **com.pinecone.orders.order-cancelled** | Order has been cancelled by customer or CSR. | Order service | Inventory (release allocation), Accounting | orderId, cancelledBy, cancelledDate, cancellationReason |
| **com.pinecone.orders.order-fulfilled** | All items have been picked, packed and shipped; the order is complete. | Fulfillment service | Accounting, Analytics, CRM | orderId, fulfillmentDate, carrier, trackingNumbers (array), packages (array with weight/dimensions), warehouseId |
| **com.pinecone.orders.order-returned** | The entire order has been returned and refunded. Individual line returns use RMA events below. | Returns service | Accounting, Inventory, CRM | orderId, returnDate, refundAmount, refundCurrency, reason (opt) |

### 4. Purchasing & Vendor Management

| Event Type | Description & Trigger | Producer | Consumers | Payload Fields |
| --- | --- | --- | --- | --- |
| **com.pinecone.purchasing.purchaseorder-created** | A new purchase order (PO) has been generated for a vendor based on reorder points or manual requisition. | Purchasing service | Vendor integration, Accounting, Inventory | poId (int), vendorId, orderDate, expectedDeliveryDate, items (array of {productId, quantity, unitPrice}), shippingMethod, paymentTerms, createdBy |
| **com.pinecone.purchasing.purchaseorder-approved** | PO has been approved by an authorised approver (based on thresholds/roles). | Purchasing service | Vendor integration, Accounting | poId, approvedBy, approvalDate, approvalNotes (opt) |
| **com.pinecone.purchasing.purchaseorder-acknowledged** | Vendor has acknowledged receipt of PO (via portal, EDI or API). | Vendor interface service | Purchasing service, Accounting | poId, vendorId, acknowledgedDate, estimatedShipDate (opt) |
| **com.pinecone.purchasing.purchaseorder-received** | Goods associated with the PO have been received (may generate multiple item‑received events). | Inventory service | Purchasing service, Accounting | poId, receivedDate, receivedBy, receiptLines (array of {productId, receivedQty}) |
| **com.pinecone.purchasing.vendor-updated** | Vendor master data has changed (e.g., contact info, lead time, payment terms). | Purchasing service | Accounts payable, PIM | vendorId, updatedFields, oldValues, newValues |

### 5. Lot & Serial Tracking

| Event Type | Description & Trigger | Producer | Consumers | Payload Fields |
| --- | --- | --- | --- | --- |
| **com.pinecone.lot.lot-created** | A new lot number has been assigned to a batch of product (manufactured or received). | Inventory service | QA/QC, Compliance, Order service | lotNumber (string), productId, quantity, manufactureDate, expirationDate (opt), supplierId (opt) |
| **com.pinecone.lot.lot-updated** | Updates to lot attributes (e.g., reclassification, extended expiry). | Inventory service | QA/QC, Compliance | lotNumber, updatedFields, oldValues, newValues |
| **com.pinecone.lot.lot-recalled** | A recall has been initiated for a lot due to quality or regulatory issues. | Compliance service | Inventory, Order, Returns | lotNumber, productId, recallDate, reason |
| **com.pinecone.serial.serial-created** | A unique serial number has been created for a tool or serialized good. | Inventory service | Warranty, Service | serialNumber, productId, manufacturer, manufactureDate |
| **com.pinecone.serial.serial-assigned** | Serial number has been assigned to a specific order or asset. | Inventory service | Order, Warranty | serialNumber, productId, assignedToOrderId (opt), assignedDate |

### 6. Shipping & Rate Shopping

| Event Type | Description & Trigger | Producer | Consumers | Payload Fields |
| --- | --- | --- | --- | --- |
| **com.pinecone.shipping.rate-calculated** | A shipping rate has been calculated for an order/package across carriers and service levels. | Shipping service | Order service, Customer portal | rateRequestId, orderId, packageDimensions (length, width, height, weight), destinationPostalCode, carrierRates (array of {carrierCode, serviceLevel, cost, estimatedDeliveryDate}) |
| **com.pinecone.shipping.label-generated** | A shipping label has been purchased and generated. | Shipping service | Fulfillment, Warehouse | shipmentId, orderId, carrier, serviceLevel, trackingNumber, labelUrl, generatedDate |
| **com.pinecone.shipping.shipment-dispatched** | Package has left the warehouse; tracking is active. | Fulfillment service | Customer portal, CRM | shipmentId, orderId, dispatchDate, carrier, trackingNumber, estimatedArrival |
| **com.pinecone.shipping.shipment-delivered** | Carrier confirms delivery of the shipment. | Carrier integration service | Order service, CRM, Accounting | shipmentId, orderId, deliveryDate, proofOfDeliveryUrl (opt) |
| **com.pinecone.shipping.hazmat-doc-uploaded** | Hazardous material shipping documentation has been uploaded/updated. | Shipping service | Compliance, Customer service | documentId, shipmentId, productId, hazmatClass, uploadDate, documentUrl |

### 7. Returns & RMA

| Event Type | Description & Trigger | Producer | Consumers | Payload Fields |
| --- | --- | --- | --- | --- |
| **com.pinecone.returns.rma-requested** | A return request (RMA) has been created by the customer or CSR. | Returns service | Inventory, Accounting, Customer service | rmaId (int), orderId, customerId, requestDate, lineItems (array of {orderLineId, returnQty, reasonCode}), requestedBy |
| **com.pinecone.returns.rma-approved** | The RMA has been approved and instructions issued. | Returns service | Warehouse, Customer service | rmaId, approvedBy, approvalDate, returnAuthorizationNumber, returnInstructions |
| **com.pinecone.returns.rma-received** | Returned items have arrived at the warehouse. | Inventory service | Returns service, Accounting | rmaId, receivedDate, receivedBy, items (array of {productId, receivedQty, condition}) |
| **com.pinecone.returns.rma-inspected** | Items have been inspected and disposition assigned. | Returns service | Inventory, Accounting | rmaId, inspectionDate, inspectedBy, items (array of {productId, dispositionCode, refundableQty}) |
| **com.pinecone.returns.rma-completed** | RMA process is complete; refund and restocking actions performed. | Returns service | Accounting, CRM | rmaId, completedDate, refundAmount, refundCurrency, dispositionSummary |

### 8. Tax & Accounting

| Event Type | Description & Trigger | Producer | Consumers | Payload Fields |
| --- | --- | --- | --- | --- |
| **com.pinecone.tax.tax-calculated** | Sales tax has been calculated for an order, invoice or line item (county/state). | Tax service | Order service, Accounting | taxCalculationId, orderId (opt), invoiceId (opt), taxJurisdiction, taxAmount, taxRate, calculationDate |
| **com.pinecone.accounting.journal-entry-created** | A journal entry has been generated for a transaction (sale, purchase, adjustment). | Accounting service | Finance, Audit | journalEntryId, transactionId (order, invoice, PO or inventory adjustment), date, entries (array of {accountCode, debit, credit}), reference (string), description (opt) |
| **com.pinecone.accounting.invoice-created** | An invoice has been issued for a customer order or vendor bill. | Accounting service | Billing, Payment service | invoiceId, orderId (opt), poId (opt), customerId or vendorId, invoiceDate, dueDate, totalAmount, currency, lineItems (array of {description, amount}) |
| **com.pinecone.accounting.payment-received** | A payment has been received from a customer or to a vendor. | Payment service | Accounting, Order service | paymentId, invoiceId (opt), poId (opt), payerId, paymentDate, paymentAmount, currency, paymentMethod, reference (opt) |

### 9. Operational Analytics & Alerts

| Event Type | Description & Trigger | Producer | Consumers | Payload Fields |
| --- | --- | --- | --- | --- |
| **com.pinecone.analytics.metric-updated** | A KPI or metric (e.g., cycle count accuracy, order fulfilment time, vendor performance) has been updated. | Analytics service | Dashboard, Alerts | metricId, metricName, value, timestamp, tags (map) |
| **com.pinecone.alerts.threshold-breached** | An operational threshold (e.g., inventory below reorder point, SLA violation) has been breached. | Analytics/Monitoring service | Email/SMS/Notification service | alertId, metricName, threshold, currentValue, severity, affectedEntityId (opt), triggeredAt, message |
| **com.pinecone.alerts.user-notified** | Notification has been sent to a user (email, SMS, portal). | Notification service | Audit, CRM | alertId, userId, notificationChannel, sentAt, deliveryStatus |

### 10. Phase 2 Enhancements (Future Events)

The following events will be introduced in Phase 2 to support demand forecasting, promotions, light manufacturing/kitting, 3PL integration, customer service console and EDI integration (as outlined in the product roadmap):

| Event Type | Description & Trigger | Producer | Consumers | Payload Fields |
| --- | --- | --- | --- | --- |
| **com.pinecone.forecasting.demand-forecast-generated** | A demand forecast has been generated for a product or product family. | Forecasting service | Purchasing, Planning, Analytics | forecastId, productId (or categoryId), forecastPeriod (start/end dates), forecastQuantity, confidenceInterval, modelVersion, generatedDate |
| **com.pinecone.promotions.promo-assigned** | A promotional price or contract pricing has been applied to a product/customer group. | Promotions service | Pricing, Order service | promotionId, productId (opt), customerGroupId (opt), discountType (percentage or amount), discountValue, startDate, endDate, usageLimit (opt) |
| **com.pinecone.manufacturing.kit-assembled** | A kit or bundle has been assembled from component SKUs. | Manufacturing/Kitting service | Inventory, Order service | kitId, productId (finished goods), components (array of {componentSku, quantity}), assembledQty, assemblyDate, warehouseId |
| **com.pinecone.3pl.sync-started** | Synchronization with a third‑party logistics provider (3PL) has started. | 3PL integration service | Inventory, Order service | syncId, providerName, startTime, syncType (inbound/outbound) |
| **com.pinecone.3pl.sync-completed** | 3PL sync has completed successfully or with errors. | 3PL integration service | Inventory, Order service, Analytics | syncId, providerName, endTime, status (Success/Failed), errorCount |
| **com.pinecone.customerservice.case-created** | A customer service case or ticket has been created. | Customer Service console | CRM, Order service, Returns | caseId, customerId, orderId (opt), subject, priority, createdDate, assignedTo (opt) |
| **com.pinecone.customerservice.case-resolved** | Customer service case resolved and closed. | Customer Service console | CRM, Analytics | caseId, resolutionDate, resolutionSummary, resolvedBy, resolutionCode |
| **com.pinecone.edi.message-sent** | An EDI document (e.g., 850 Purchase Order, 810 Invoice) has been sent to or from a trading partner. | EDI gateway | Purchasing, Accounting, Logistics | ediMessageId, documentType, partnerId, sentDate, status, documentUrl |
| **com.pinecone.edi.message-received** | An EDI message has been received and parsed. | EDI gateway | Purchasing, Accounting, Inventory | ediMessageId, documentType, partnerId, receivedDate, status, parsedData (object) |

## Message Contracts

All events are serialized as **JSON** objects using UTF‑8 encoding and published over the selected message broker (e.g., Azure Service Bus topics). Each message must include:

1. **Headers / Transport metadata** — Provided by the broker, such as topic name, partition key and correlation ID. For CloudEvents integration, map the metadata fields to the appropriate header attributes (e.g., ce-id for eventId, ce-source for source, ce-type for eventType, ce-specversion for specversion, ce-time for timestamp).
2. **Body** — The JSON envelope containing the metadata and data objects as described above.
3. **Schema registry or versioning** — For strongly typed contracts, store JSON schemas in a schema registry (e.g., Azure Schema Registry, Confluent Schema Registry). Consumers should validate events against the correct schema version before processing. Schema URIs can be included in the dataschema attribute【280384752129849†L456-L463】.

## Reporting & Audit Considerations

Each event provides a durable audit trail of significant business actions. Persist events in an event store or append‑only log to support regulatory reporting, debugging and analytics. Include user identifiers, timestamps and reasons in the payload to facilitate accountability. Consumers should log receipt and processing outcomes for each event. Use correlation IDs to trace end‑to‑end flows and reconstruct sagas across services【806325492643920†L214-L224】.

## Conclusion

This document standardizes how PineCone Pro Supplies defines, names and publishes events across its microservices. By following these conventions—clear naming, rich metadata, versioning, idempotence handling and secure payload design—the system achieves reliable, traceable and scalable event‑driven communication. Future services can extend the event catalog while maintaining compatibility by adhering to the same guidelines.