International Standard

ISO/IEC 19987

Third edition
2024-03

Information technology — EPC Information Services (EPCIS)

Technologies de l'information — Services d'information sur les codes de produit électronique

Reference number
ISO/IEC 19987:2024(en)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

ISO and IEC draw attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO and IEC take no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO and IEC had not received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents and https://patents.iec.ch. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by GS1 (as EPCIS Standard, Release 2.0) and drafted in accordance with its editorial rules. It was adopted, under the JTC 1 PAS procedure, by Joint Technical Committee ISO/IEC JTC 1, *Information technology*.

This third edition cancels and replaces the second edition (ISO/IEC 19987:2017), which has been technically revised.

The main changes are as follows:

— addition of JSON/SON-LD syntax (alongside XML);

— addition of REST bindings (alongside SOAP/WSDL);

— complete overhaul of UML diagrams;

— clarification on distinction between standard vocabulary and user vocabulary;

— new AssociationEvent;

— new “How” event dimension;

— overview of EPCIS even “dimensions” with cross-references to relevant sections in EPCIS (this document) and CBV (ISO/IEC 19988);

— new Persistent Disposition indicating non-transient business state of an object;

— new SensorElement to accommodate sensor data;

— addition of certificationInfo to core EPCISEvent;

— update of SimpleEventQuery parameters;

— removal of support for Simple Master Data Query and EPCIS Master Data Document.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

Table of Contents

Index of figures

1 Introduction

This document is a GS1 standard that defines Version 2.0 of EPC Information Services (EPCIS). The goal of EPCIS is to enable disparate applications to create and share visibility event data, both within and across enterprises. Ultimately, this sharing is aimed at enabling users to gain a shared view of physical or digital objects within a relevant business context.

"Objects" in the context of EPCIS typically refers to physical objects that are identified either at a class or instance level and which are handled in physical handling steps of an overall business process involving one or more organisations. Examples of such physical objects include trade items (products), logistic units, returnable assets, fixed assets, physical documents, etc. "Objects" may also refer to digital objects, also identified at either a class or instance level, which participate in comparable business process steps. Examples of such digital objects include digital trade items (music downloads, electronic books, etc.), digital documents (electronic coupons, etc.), and so forth. Throughout this document the word "object" is used to denote a physical or digital object, identified at a class or instance level, that is the subject of a business process step. EPCIS data consist of "visibility events," each of which is the record of the completion of a specific business process step acting upon one or more objects.

The EPCIS standard was originally conceived as part of a broader effort to enhance collaboration between trading partners by sharing of detailed information about physical or digital objects. The name EPCIS reflects the origins of this effort in the development of the Electronic Product Code (EPC). It should be noted, however, that EPCIS does not require the use of Electronic Product Codes, nor of Radio-Frequency Identification (RFID) data carriers, and does not even require instance-level identification (for which the Electronic Product Code was originally designed). The EPCIS standard applies to all situations in which visibility event data is to be captured and shared, and the presence of "EPC" within the name is of historical significance only.

EPCIS provides open, standardised interfaces that allow for seamless integration of well-defined services in inter-company environments as well as within companies. Standard interfaces are defined in the EPCIS standard to enable visibility event data to be captured and queried using a defined set of service operations and associated data standards, all combined with appropriate security mechanisms that satisfy the needs of user companies. In many or most cases, this will involve the use of one or more persistent databases of visibility event data, though elements of the Services approach could be used for direct application-to-application sharing without persistent databases.

With or without persistent databases, the EPCIS specification specifies only standard data sharing interfaces between applications that capture visibility event data and those that need access to it. *It does not specify how the service operations or databases themselves should be implemented.* This includes not defining how the EPCIS services should acquire and/or compute the data they need, except to the extent the data is captured using the standard EPCIS capture operations. The interfaces are needed for interoperability, while the implementations allow for competition among those providing the technology and implementing the standard.

EPCIS is intended to be used in conjunction with the GS1 Core Business Vocabulary (CBV) standard [CBV2.0]. EPCIS and the CBV are developed, maintained and published by GS1; EPCIS and the CBV are also published within ISO's PAS process as ISO/IEC 19987 and ISO/IEC 19988, respectively. The CBV standard provides definitions of data values that may be used to populate the data structures defined in the EPCIS standard. The use of the standardised vocabulary provided by the CBV standard is critical to interoperability and critical to provide for querying of data by reducing the variation in how different businesses express common intent. Therefore, applications should use the CBV standard to the greatest extent possible in constructing EPCIS data.

The companion EPCIS and CBV Implementation Guideline [EPCISGuideline] provides additional guidance for building visibility systems using EPCIS and CBV, including detailed discussion of how to model specific business situations using EPCIS/CBV data and methods for sharing such data between trading partners.

2 Relationship to the GS1 System Architecture

This section is largely quoted from [GS1Arch], and shows the relationship of EPCIS to other GS1 standards.

2.1 Overview of GS1 standards

GS1 standards support the information needs of end users interacting with each other in supply chains, specifically the information required to support the business processes through which supply chain participants interact. The subjects of such information are the real-world entities that are part of those business processes. Real-world entities include things traded between companies, such as products, parts, raw materials, packaging, and so on. Other real-world entities of relevance to trading partners include the equipment and material needed to carry out the business processes surrounding trade such as containers, transport, machinery; entities corresponding to physical locations in which the business processes are carried out; legal entities such as companies, divisions; service relationships; business transactions and documents; and others. Real-world entities may exist in the tangible world, or may be digital or conceptual. Examples of physical objects include a consumer electronics product, a transport container, and a manufacturing site (location entity). Examples of digital objects include an electronic music download, an eBook, and an electronic coupon. Examples of conceptual entities include a trade item class, a product category, and a legal entity.

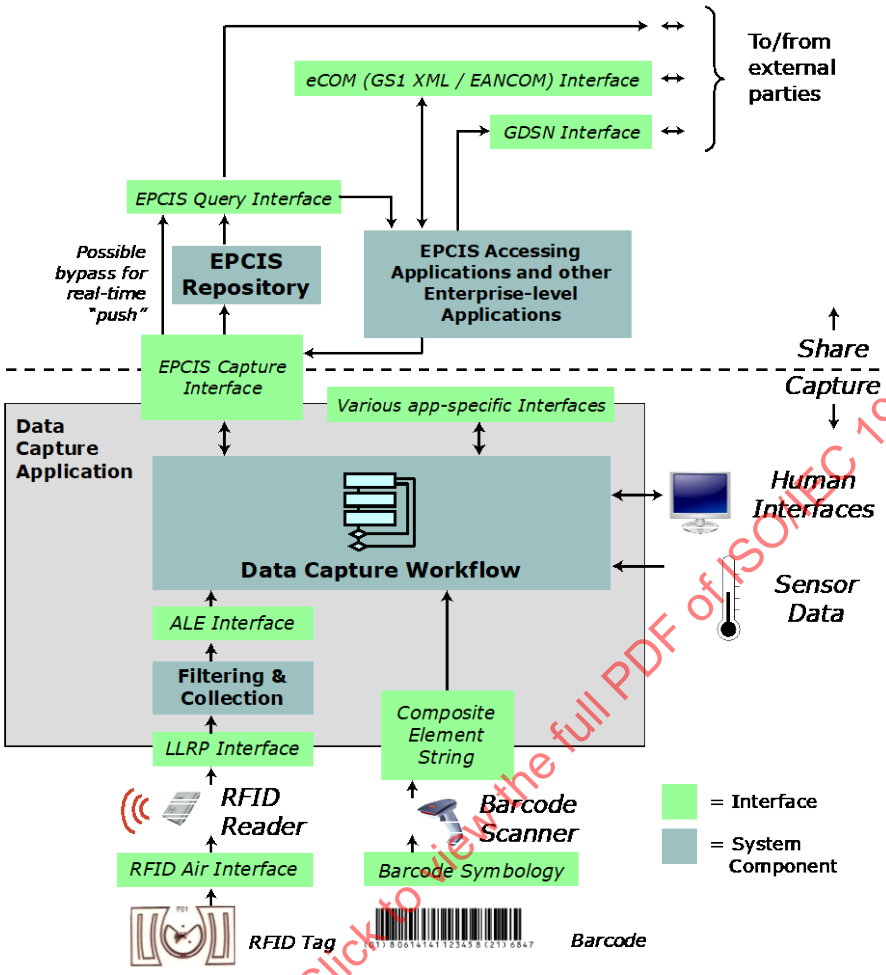
GS1 standards may be divided into the following groups according to their role in supporting information needs related to real-world entities in supply chain business processes:

- Standards which provide the means to **identify** real-world entities so that they may be the subject of electronic information that is stored and/or communicated by end users. GS1 identification standards include standards that define unique identification codes (called GS1 identification keys).
- Standards which provide the means to automatically **capture** data that is carried directly on physical objects, bridging the world of physical things and the world of electronic information. GS1 data capture standards include definitions of barcode and radio-frequency identification (RFID) data carriers which allow identifiers to be affixed directly to a physical object, and standards that specify consistent interfaces to readers, printers, and other hardware and software components that connect the data carriers to business applications.
- Standards which provide the means to **Share** information, both between trading partners and internally, providing the foundation for electronic business transactions, electronic visibility of the physical or digital world, and other information applications. GS1 standards for information sharing include this EPCIS Standard which is a standard for visibility event data. Other standards in the "Share" group are standards for master data and for business transaction data, as well as discovery standards that help locate where relevant data resides across a supply chain and trust standards that help establish the conditions for sharing data with adequate security.

The EPCIS standard fits into the "Share" group, providing the data standard for visibility event data and the interface standards for capturing such information from data capture infrastructure (which employs standards from the "Capture" group) and for sharing such information with business applications and with trading partners.

2.2 EPCIS in relation to the "Capture" and "Share" layers

Figure 2-1 EPCIS in relation to the "Capture" and "Share" layers



The diagram above shows the relationship between EPCIS and other GS1 standards in the "Capture" and "Share" groups. (The "Identify" group of standards pervades the data at all levels of this architecture, and so is not explicitly shown.)

As depicted in the diagram above, the EPCIS Capture Interface exists as a bridge between the "Capture" and "Share" standards. The EPCIS Query Interface provides visibility event data both to internal applications and for sharing with trading partners.

At the centre of a data capture application is the data capture workflow that supervises the business process step within which data capture takes place. This is typically custom logic that is specific to the application. Beneath the data capture workflow in the diagram is the data path between the workflow and GS1 data carriers: barcodes and RFID. The green bars in the diagram denote GS1 standards that may be used as interfaces to the data carriers. At the top of the diagram are the interfaces between the data capture workflow and larger-scale enterprise applications. Many of these interfaces are application- or enterprise-specific, though using GS1 data as building blocks; however, the EPCIS interface is a GS1 standard. Note that the interfaces at the top of the diagram, including EPCIS, are independent of the data carrier used at the bottom of the diagram.

The purpose of the interfaces and the reason for a multi-layer data capture architecture is to provide isolation between different levels of abstraction. Viewed from the perspective of an enterprise application (i.e., from the uppermost blue box in the figure), the entire data capture application shields the enterprise application from the details of exactly how data capture takes place. Through the application-level interfaces (uppermost green bars), an enterprise application interacts with the data capture workflow through data that is data carrier independent and in which all of the interaction between data capture components has been consolidated into that data. At a lower level, the data capture workflow is cognizant of whether it is interacting with

barcode scanners, RFID interrogators, human input, etc., but the transfer interfaces (green bars in the middle) shield the data capture workflow from low-level hardware details of exactly how the data carriers work. The lowest level interfaces (green bars on the bottom) embody those internal data carrier details. EPCIS and the "Share" layer in general differ from elements in the Capture layer in three key respects:

1. EPCIS deals explicitly with historical data (in addition to current data). The Capture layer, in contrast, is oriented exclusively towards real-time processing of captured data.
2. EPCIS often deals not just with raw data captured from data carriers such as barcodes and RFID tags, but also in contexts that imbue those observations with meaning relative to the physical or digital world and to specific steps in operational or analytical business processes. The Capture layers are more purely observational in nature. An EPCIS event, while containing much of the same "Identify" data as a Filtering & Collection event or a barcode scan, is at a semantically higher level because it incorporates an understanding of the business context in which the identifier data were obtained. Moreover, there is no requirement that an EPCIS event be directly related to a specific physical data carrier observation. For example, an EPCIS event may indicate that a perishable trade item has just crossed its expiration date; such an event may be generated purely by software.
3. EPCIS operates within enterprise IT environments at a level that is much more diverse and multi-purpose than exists at the Capture layer, where typically systems are self-contained and exist to serve a single business purpose. In part, and most importantly, this is due to the desire to share EPCIS data between enterprises which are likely to have different solutions deployed to perform similar tasks. In part, it is also due to the persistent nature of EPCIS data. And lastly, it is due to EPCIS being at the highest level of the overall architecture, and hence the natural point of entry into other enterprise systems, which vary widely from one enterprise to the next (or even within parts of the same enterprise).

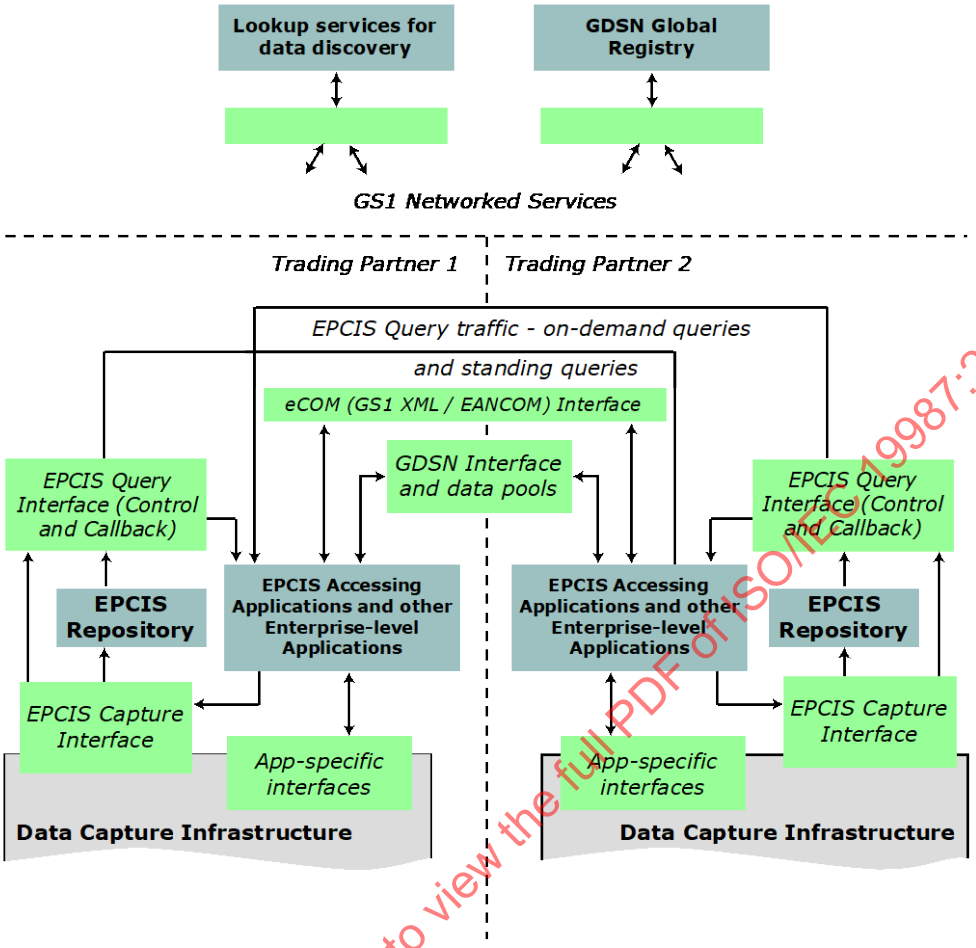
2.3 EPCIS in Relation to trading partners

GS1 standards in the "Share" layer pertain to three categories of data that are shared between end users:

Data	Description	GS1 standards
Master data	Data, shared by one trading partner to many trading partners, that provide descriptive attributes of real-world entities identified by GS1 identification keys, including trade items, parties, and physical locations.	GDSN
Transaction data	Trade transactions triggering or confirming the execution of a function within a business process as defined by an explicit business agreement (e.g., a supply contract) or an implicit one (e.g., customs processing), from the start of the business process (e.g., ordering the product) to the end of it (e.g., final settlement), also making use of GS1 identification keys.	GS1 XML EANCOM
Visibility event data	Details about physical or digital activity in the supply chain of products and other assets, identified by keys, detailing where these objects are in time, and why; not just within one organisation's four walls, but across organisations.	EPCIS

Transaction Data and Visibility Event Data have the characteristic that new documents of those types are continually created as more business is transacted in a supply chain in steady state, even if no new real-world entities are being created. Master data, in contrast, is more static: the master data for a given entity changes very slowly (if at all), and the quantity of master data only increases as new entities are created, not merely because existing entities participate in business processes. For example, as a given trade item instance moves through the supply chain, new transaction data and visibility event data are generated as that instance undergoes business transactions (such as purchase and sale) and physical handling processes (packing, picking, stocking, etc.). But new master data is only created when a new trade item or location is added to the supply chain.

Figure 2-2 EPCIS in relation to other GS1 System Architecture components



The figure above illustrates the flow of data between trading partners, emphasising the parts of the EPCIS standard involved in the flow of visibility event data.

In addition to the use of the EPCIS Query Interface as illustrated above, trading partners may by mutual agreement use the EPCIS Document structure defined in section 9.3 as a means to transport a collection of EPCIS events, optionally accompanied by relevant master data, as a single electronic document.

2.4 EPCIS in relation to other GS1 System Architecture components

The following outlines the responsibilities of each element of the GS1 System Architecture as illustrated in the figures in the preceding sections. Further information may be found in [GS1Arch].

- *RFID and Barcode Readers* Make observations of RFID tags while they are in the read zone, and observations of barcodes when reading is triggered.
- *Low-Level [RFID] Reader Protocol (LLRP) Interface* Defines the control and delivery of raw RFID tag reads from RFID Readers to the Filtering & Collection role. Events at this interface say "Reader A saw EPC X at time T."
- *Filtering & Collection* This role filters and collects raw RFID tag reads, over time intervals delimited by events defined by the EPCIS Capturing Application (e.g. tripping a motion detector). No comparable role typically exists for reading barcodes, because barcode readers typically only read a single barcode when triggered.
- *Filtering & Collection (ALE) Interface* Defines the control and delivery of filtered and collected RFID tag read data from the Filtering & Collection role to the Data Capture Workflow role. Events at this interface say "At Logical Reader L, between time T1 and T2, the following EPCs were observed," where the list of EPCs has no

duplicates and has been filtered by criteria defined by the EPCIS Capturing Application. In the case of barcodes, comparable data is delivered to the Data Capture Workflow role directly from the barcode reader in the form of a GS1 Element String.

- *Data Capture Workflow* Supervises the operation of the lower-level architectural elements, and provides business context by coordinating with other sources of information involved in executing a particular step of a business process. The Data Capture Workflow may, for example, coordinate a conveyor system with Filtering & Collection events and barcode reads, may check for exceptional conditions and take corrective action (e.g., diverting a bad object into a rework area), may present information to a human operator, and so on. The Data Capture Workflow understands the business process step or steps during which EPCIS event data capture takes place. This role may be complex, involving the association of multiple Filtering & Collection events and/or barcode reads with one or more business events, as in the loading of a shipment. Or it may be straightforward, as in an inventory business process where there may be readers deployed that generate observations about objects that enter or leave the shelf. Here the Filtering & Collection-level event or barcode read and the EPCIS-level event may be so similar that very little actual processing at the Data Capture Workflow level is necessary, and the Data Capture Workflow merely configures and routes events from the Filtering & Collection interface and/or barcode readers directly through the EPCIS Capture Interface to an EPCIS Repository or a business application. A Data Capture Workflow whose primary output consists of EPCIS events is called an "EPCIS Capturing Application" within this standard.
- *EPCIS Interfaces* The interfaces through which EPCIS data is delivered to enterprise-level roles, including EPCIS Repositories, EPCIS Accessing Applications, and data exchange with partners. Events at these interfaces say, for example, "At location X, at time T, the following contained objects (cases) were verified as being aggregated to the following containing object (pallet)." **There are three EPCIS Interfaces**, specified normatively in this document:
  - The **EPCIS Capture Interface** defines the delivery of EPCIS events from EPCIS Capturing Applications to other roles that consume the data in real time, including EPCIS Repositories, and real-time "push" to EPCIS Accessing Applications and trading partners.
  - The **EPCIS Query Control Interface** defines a means for EPCIS Accessing Applications and trading partners to obtain EPCIS data subsequent to capture, typically by interacting with an EPCIS Repository. The EPCIS Query Control Interface provides two modes of interaction. In "on-demand" or "synchronous" mode, a client makes a request through the EPCIS Query Control Interface and receives a response immediately. In "standing request" or "asynchronous" mode, a client establishes a subscription for a periodic query.
  - Each time the periodic query is executed, the results are delivered asynchronously (or "pushed") to a recipient via the **EPCIS Query Callback Interface**. The EPCIS Query Callback Interface may also be used to deliver information immediately upon capture; this corresponds to the "possible bypass for real-time push" arrow in the diagram.
- *EPCIS Accessing Application:* Responsible for carrying out overall enterprise business processes, such as warehouse management, shipping and receiving, historical throughput analysis, and so forth, aided by EPCIS visibility event data.
- *EPCIS Repository:* Records EPCIS-level events generated by one or more EPCIS Capturing Applications and makes them available for later query by EPCIS Accessing Applications.
- *Partner Application:* Trading Partner systems that perform the same role as an EPCIS Accessing Application, though from outside the responding party's network. Partner Applications may be granted access to a subset of the information that is available from an EPCIS Capturing Application or within an EPCIS Repository.

The interfaces within this stack are designed to insulate the higher levels of the architecture from unnecessary details of how the lower levels are implemented. One way to understand this is to consider what happens if certain changes are made:

- The *Low-Level [RFID] Reader Protocol (LLRP) and GS1 Element String* insulate the higher layers from knowing what RF protocols or barcode symbologies are in use, and what reader makes/models have been chosen. If a different reader is substituted, the information sent through these interfaces remains the same.
- In situations where RFID is used, the Filtering & Collection Interface insulates the higher layers from the physical design choices made regarding how RFID tags are sensed and accumulated, and how the time boundaries of events are triggered. If a single four-antenna RFID reader is replaced by a constellation of five single-antenna "smart antenna" readers, the events at the Filtering & Collection level remain the same. Likewise, if a different triggering mechanism is used to mark the start and end of the time interval over which reads are accumulated, the Filtering & Collection event remains the same.
- EPCIS insulates enterprise applications from understanding the details of how individual steps in a business process are carried out at a detailed level. For example, a typical EPCIS event is "At location X, at time T, the following cases were verified as being on the following pallet." In a conveyor-based business implementation, this may correspond to a single Filtering & Collection event, in which reads are accumulated during a time interval whose start and end is triggered by the case crossing electric eyes surrounding a reader mounted on the conveyor. But another implementation could involve three strong people who move around the cases and use hand-held readers to read the tags. At the Filtering & Collection level, this looks very different (each triggering of the hand-held reader is likely a distinct Filtering & Collection event), and the processing done by the EPCIS Capturing Application is quite different (perhaps involving an interactive console that the people use to verify their work). But the EPCIS event is still the same for all these implementations.

In summary, EPCIS-level data differs from data employed at the Capture level in the GS1 System Architecture by incorporating semantic information about the business process in which data is collected, and providing historical observations. In doing so, EPCIS insulates applications that consume this information from knowing the low-level details of exactly how a given business process step is carried out.

3 EPCIS specification principles

The considerations in the previous two sections reveal that the requirements for standards at the EPCIS layer are considerably more complex than in the Capture layer of the GS1 System Architecture. The historical nature of EPCIS data implies that EPCIS interfaces need a richer set of access techniques than ALE or RFID and barcode reader interfaces. The incorporation of operational or business process context into EPCIS implies that EPCIS traffics in a richer set of data types, and moreover needs to be much more open to extension in order to accommodate the wide variety of business processes in the world. Finally, the diverse environment in which EPCIS operates implies that the EPCIS Standard be layered carefully so that even when EPCIS is used between external systems that differ widely in their details of operation, there is consistency and interoperability at the level of what the abstract structure of the data is and what the data means.

In response to these requirements, EPCIS is described by a framework specification and narrower, more detailed specifications that populate that framework. The framework is designed to be:

- *Layered:* In particular, the structure and meaning of data in an abstract sense is specified separately from the concrete details of data access services and bindings to particular interface protocols. This allows for variation in the concrete details over time and across enterprises while preserving a common meaning of the data itself. It also permits EPCIS data specifications to be reused in approaches other than the service-oriented approach of the present specification. For example, data definitions could be reused in an EDI framework.
- *Extensible:* The core specifications provide a core set of data types and operations, but also provide several means whereby the core set may be extended for purposes specific to a given industry or application area. Extensions not only provide for proprietary requirements to be addressed in a way that leverages as much of the standard framework as possible, but also provides a natural path for the standards to evolve and grow over time.
- *Modular:* The layering and extensibility mechanisms allow different parts of the complete EPCIS framework to be specified by different documents, while promoting coherence across the entire framework. This allows the process of standardisation (as well as of implementation) to scale.

The remainder of this document specifies the EPCIS framework. It also populates that framework with a core set of data types and data interfaces. The companion standard, the GS1 Core Business Vocabulary (CBV), provides additional data definitions that layer on top of what is provided by the EPCIS standard.

4 Terminology and typographical conventions

Within this specification, the terms SHALL, SHALL NOT, SHOULD, SHOULD NOT, MAY, CAN, and CANNOT are to be interpreted as specified in section 7 ("*Verbal forms for expressions of provisions*") of the ISO/IEC Directives, Part 2, 2018, 8th edition [ISODir2]. When used in this way, these terms will always be shown in ALL CAPS; when these words appear in ordinary typeface they are intended to have their ordinary English meaning.

All sections of this document, are normative, except where explicitly noted as non-normative.

The following typographical conventions are used throughout the document:

- ALL CAPS type is used for the special terms from [ISODir2] enumerated above.
- `Monospace` type is used to denote programming language, UML, XML, and JSON (or JSON-LD) identifiers, as well as for the text of XML and JSON (or JSON-LD) documents.

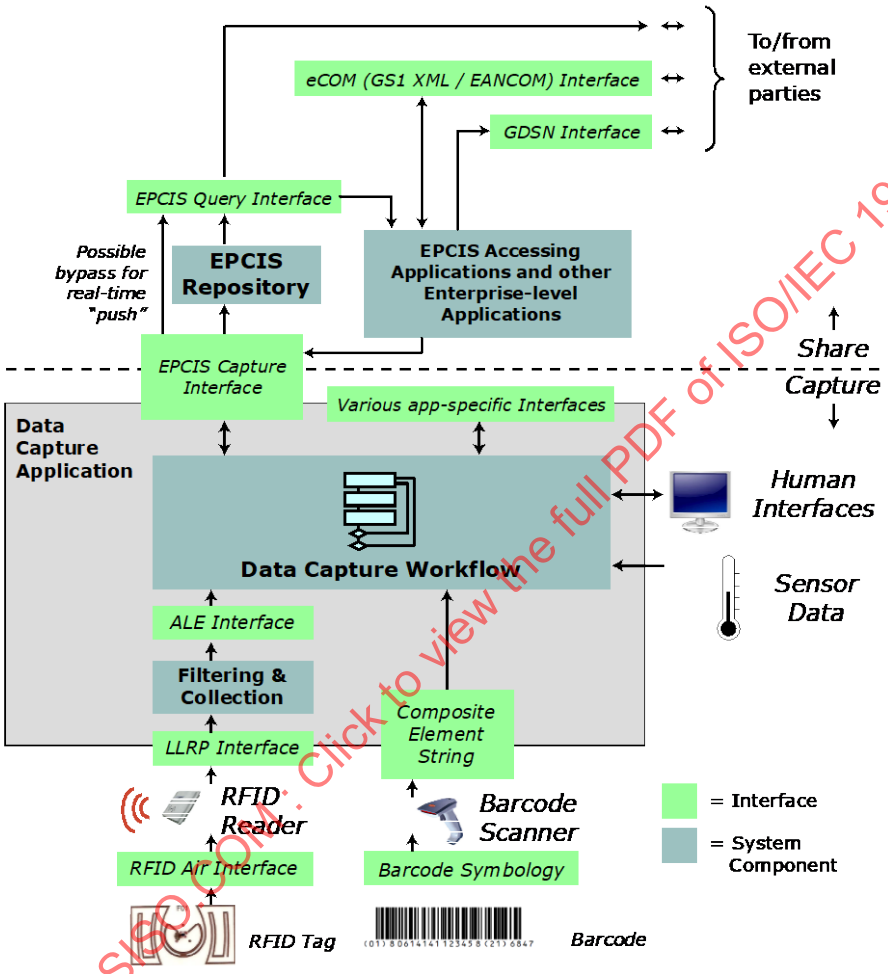
5 EPCIS specification framework

The EPCIS specification is designed to be layered, extensible, and modular.

5.1 Layers

The EPCIS specification framework is organised into several layers, as illustrated in the diagram and described below.

Figure 5-1 Layers of the EPCIS specification framework



- *Abstract Data Model Layer:* The Abstract Data Model Layer specifies the generic structure of EPCIS data. This is the only layer that is not extensible by mechanisms other than a revision to the EPCIS specification itself. The Abstract Data Model Layer specifies the general requirements for creating data definitions within the Data Definition Layer.
- *Data Definition Layer:* The Data Definition Layer specifies what data is exchanged through EPCIS, what its abstract structure is, and what it means. **One data definition module** is defined within the present specification, called **the Core Event Types Module**. Data definitions in the Data Definition Layer are specified abstractly, following rules defined by the Abstract Data Model Layer.
- *Service Layer:* The Service Layer defines service interfaces through which EPCIS clients interact. In the present specification, **two service layer modules** are defined.
  - The Core **Capture** Operations Module defines one service interface (the EPCIS **Capture Interface**) through which EPCIS Capturing Applications use to deliver Core Event Types to interested parties.
  - The Core **Query** Operations Module defines two service interfaces (the EPCIS **Query Control Interface** and the EPCIS **Query Callback Interface**) that

EPCIS Accessing Applications use to obtain data previously captured. Interface definitions in the Service Layer are specified abstractly using UML.

- *Bindings:* Bindings specify concrete realisations of the Data Definition Layer and the Service Layer. There may be many bindings defined for any given Data Definition or Service module. In this specification, a number of bindings are specified for the three modules defined in the Data Definition and Service Layers.
  - The data definitions in the **Core Event Types** data definition module are given bindings to XML schema, JSON schema and SHACL.
  - The EPCIS **Capture** Interface in the Core Capture Operations Module is given bindings for Message Queue, HTTP and REST.
  - The EPCIS **Query Control** Interface in the Core Query Operations Module is given a binding to SOAP over HTTP via a WSDL web services description, a binding for AS2, and a REST binding.
  - The EPCIS **Query Callback** Interface in the Core Query Operations Module is given bindings to HTTP, HTTPS, and AS2; the REST binding specifies the **WebSocket** to support subscriptions.
- *GS1 Core Business Vocabulary Standard:* The GS1 Core Business Vocabulary standard [CBV] is a companion to the EPCIS standard. It defines specific vocabulary elements that may be used to populate the data definitions specified in the Data Definition Layer of the EPCIS standard. While EPCIS may be used without CBV, by employing only private or proprietary data values, it is far more beneficial for EPCIS applications to make as much use of the CBV Standard as possible.

5.2 Extensibility

The layered technique for specification promotes extensibility, as one layer may be reused by more than one implementation in another layer. For example, while this specification includes an XML binding of the Core Event Types data definition module, another specification may define a binding of the same module to a different syntax, for example a CSV file.

Besides the extensibility inherent in layering, the EPCIS specification includes several specific mechanisms for extensibility:

- *Subclassing:* Data definitions in the Data Definition Layer are defined using UML, which allows a new data definition to be introduced by creating a subclass of an existing one. A subclass is a new type that includes all of the fields of an existing type, extending it with new fields. An instance of a subclass may be used in any context in which an instance of the parent class is expected.
- *Extension Points:* Data definitions and service specifications also include extension points which vendors may use to provide extended functionality without creating subclasses.

5.3 Modularity

The EPCIS specification framework is designed to be modular. That is, it does not consist of a single specification, but rather a collection of individual specifications that are interrelated. This allows EPCIS to grow and evolve in a distributed fashion. The layered structure and the extension mechanisms provide the essential ingredients to achieving modularity, as does the grouping into modules.

While EPCIS specifications are modular, there is no requirement that the module boundaries of the specifications be visible or explicit within *implementations* of EPCIS. For example, there may be a particular software product that provides a SOAP/HTTP-based implementation of a case-to-pallet association service and a product catalogue service that traffics in data defined in the relevant data definition modules. This product may conform to as many as six different modules from the EPCIS standard: the data definition module that describes product catalogue data, the data definition module that defines case-to-pallet associations, the specifications for the respective services, and the respective SOAP/HTTP bindings. But the source code of the product may have no trace of these boundaries, and indeed the concrete database schema used by the product may denormalise the data so that product catalogue and case-to-

pallet association data are inextricably entwined. But as long as the net result conforms to the specifications, this implementation is permitted.

6 Abstract data model layer

This section gives a normative description of the abstract data model that underlies EPCIS.

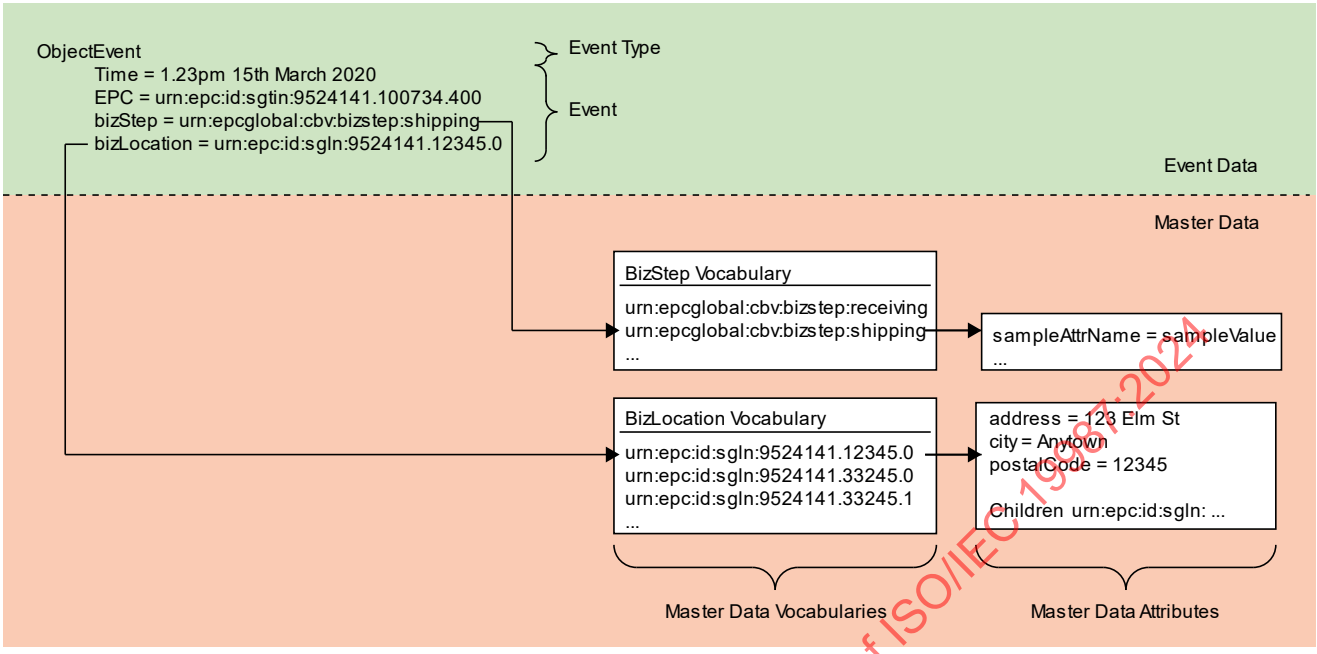
6.1 Event data and master data

Generically, EPCIS deals in two kinds of data: event data and master data. Event data arises in the course of carrying out business processes, and is captured through the EPCIS Capture Interface and made available for query through the EPCIS Query Interfaces. Master data is additional data that provides the necessary context for interpreting the event data. It is available for query through the EPCIS Query Control Interface, but the means by which master data enters the system is not specified in the EPCIS standard.

The Abstract Data Model Layer does not attempt to define the meaning of the terms "event data" or "master data," other than to provide precise definitions of the structure of the data as used by the EPCIS specification. The modelling of real-world business information as event data and master data is the responsibility of the Data Definition Layer, and of industry and end-user agreements that build on top of this specification.

Non-Normative: Explanation: While for the purposes of this specification the terms "event data" and "master data" mean nothing more than "data that fits the structure provided here," the structures defined in the Abstract Data Model Layer are designed to provide an appropriate representation for data commonly requiring exchange through EPCIS. Informally, these two types of data may be understood as follows. Event data grows in quantity as more business is transacted, and refers to things that happen at specific moments in time. An example of event data is "At 1:23pm on 15 March 2004, EPC X was observed at Location L." Master data does not generally grow merely because more business is transacted (though master data does tend to grow as organisations grow in size), is not typically tied to specific moments in time (though master data may change slowly over time), and provides interpretation for elements of event data. An example of master data is "Location L refers to the distribution centre located at 123 Elm Street, Anytown, US." All of the data in the set of use cases considered in the creation of the EPCIS standard can be modelled as a combination of event data and master data of this kind.

Figure 6-1 Structure of event data and master data in EPCIS



The structure of event data and master data in EPCIS is illustrated above. (Note that this is an illustration only: the specific vocabulary elements and master data attribute names in this figure are not defined within this specification.)

The ingredients of the EPCIS Abstract Data Model are defined below:

- *Event Data:* A set of Events.
- *Event:* A structure consisting of an Event Type and one or more named Event Fields.
- *Event Type:* A namespace-qualified name (qname) that indicates to which of several possible Event structures (as defined by the Data Definition Layer) a given event conforms.
- *Event Field:* A named field within an Event. The name of the field is given by a qname, referring either to a field name specified by the Data Definition Layer or a field name defined as an extension to this specification. The value of the field may be a primitive type (such as an integer or timestamp), a Vocabulary Element, or a list of primitive types or Vocabulary Elements.
- *Master data:* A set of Vocabularies, together with master data attributes associated with elements of those Vocabularies.
- *Vocabulary:* A named set of identifiers. The name of a Vocabulary is a qname that may be used as a type name for an event field. The identifiers within a Vocabulary are called Vocabulary Elements. A Vocabulary represents a set of alternative values that may appear as the values of specific Event Fields. Vocabularies in EPCIS are used to model sets such as the set of available location names, the set of available business process step names, and so on.
- *Vocabulary Element:* An identifier that names one of the alternatives modelled by a Vocabulary. The value of an Event Field may be a Vocabulary Element. Vocabulary Elements are represented as Uniform Resource Identifiers (URIs). Each Vocabulary Element may have associated master data attributes.
- *Master data attributes:* An unordered set of name/value pairs associated with an individual Vocabulary Element. The name part of a pair is a qname. The value part of a pair may be a value of arbitrary type. A special attribute is a (possibly empty) list of children, each child being another vocabulary element from the same vocabulary. See section 6.5.

New EPCIS Events are generated at the edge and delivered into EPCIS infrastructure through the EPCIS Capture Interface, where they can subsequently be delivered to interested applications through the EPCIS Query Interfaces. There is no mechanism provided in either interface by which an application can delete or modify an EPCIS Event. The only way to "retract" or "correct" an EPCIS Event is to generate a

subsequent event whose business meaning is to rescind or amend the effect of a prior event (section 7.4.1.2.1 discusses how this may be done).

While the EPCIS Capture Interface and EPCIS Query Interfaces provide no means for an application to explicitly request the deletion of an event, EPCIS Repositories MAY implement data retention policies that cause old EPCIS events to become inaccessible after some period of time.

Master data, in contrast, may change over time, though such changes are expected to be infrequent relative to the rate at which new event data is generated. The current version of this specification does not specify how master data changes (nor, as noted above, does it specify how master data is entered in the first place).

6.1.1 Transmission of master data in EPCIS

The EPCIS Capture and Query Interfaces are primarily concerned with the transmission of EPCIS Events. The means by which master data enters a system that implements these interfaces is not specified in the EPCIS standard. However, the EPCIS standard does provide mechanisms for transmission of master data, which an implementation may use to ensure that the recipient of EPCIS event data has access to the master data necessary to interpret that event data. Alternatively, master data may be transmitted by means entirely outside the EPCIS standard. The EPCIS standard does not impose any requirements on whether EPCIS event data is accompanied by master data or not, other than to require that master data accompanying event data be consistent with any master data in ILMD sections of those events.

The EPCIS standard provides two mechanisms for transmission of master data, summarised in the table below:

Table 6-1 Transmission of master data in EPCIS

Mechanism	Section	Description	Constraint
ILMD	7.3.8	An EPCIS event that marks the beginning of life for an instance-level or lot-level identifier may include corresponding master data directly in the event.	The master data in the event SHALL reflect the current values of master data attributes, as known to the event creator, as of the event time. Note that because this data is embedded directly in the event, it is permanently a part of that event and will always be included when this event is queried for (subject to redaction as specified in section 8.2.2).
Header of XML EPCIS document	9.5	An EPCIS document used for point-to-point transmission of a collection of EPCIS events outside of the EPCIS Query Interface may include relevant master data in the document header.	The master data in the document header SHALL reflect the current values of master data attributes, as known to the document creator, as of the time the document is created. Master data in the header of an EPCIS document SHALL NOT specify attribute values that conflict with the ILMD section of any event contained within the EPCIS document body.

6.2 Standard vocabulary and user vocabulary

Before version 2.0, EPCIS / CBV have previously used the terms 'vocabulary type' to refer to standard EPCIS event data fields whose value is expected to be populated either using one of the enumerated values of the corresponding 'standard vocabulary' defined within CBV or with a value 'user vocabulary' that might be defined by a single organisation, solution provider, industry sector or other consortium.

Starting with version 2.0, EPCIS and CBV add support for Linked Data (in particular, a JSON-LD representation and formal Linked Data ontologies for the data models). A Linked Data ontology is sometimes also referred to as a vocabulary. However, when specialists in ontologies, Linked Data or Semantic Web talk about a vocabulary, they are not only referring to enumerated code lists; a Linked Data vocabulary can also define classes, properties and identifiers in addition to code lists of enumerated

values. The table below provides a mapping of the historical terminology used within EPCIS/CBV versus the terminology used by ontologists, in which 'vocabulary' has a much broader meaning than 'enumeration'.

Table 6-2 Mapping of terminology used in EPCIS/CBV and by ontologists

Example	EPCIS/CBV describes as	Ontological description (A Linked Data Vocabulary or Web Vocabulary can define anything below)
`ObjectEvent`	Event Type	Class
`eventTime` field	A field	Datatype Property (expects a non-URI literal value)
`epcList` field	A field	Object Property (expecting a URI)
`bizStep` field	A 'Vocabulary Type'	Object Property (expecting a URI)
'`shipping`'	An individual value within CBV BizStep Vocabulary	Individual within a Class (value within a code list enumeration)
['`shipping`', '`receiving`', …]	a CBV Vocabulary, 'BizStep'	Class containing individuals, each individual having a definition, the class representing a code list

Vocabularies are used extensively within EPCIS to model physical, digital, and conceptual entities that exist in the real world. Examples of vocabularies defined in the core EPCIS Data Definition Layer are location names, object class names (an object class name is something like "Acme Deluxe Widget," as opposed to an EPC which names a specific instance of an Acme Deluxe Widget), and business step names. In each case, a vocabulary represents a finite (though open-ended) set of alternatives that may appear in specific fields of events.

It is useful to distinguish two kinds of vocabularies, which follow different patterns in the way they are defined and extended over time:

- *Standard Vocabulary:* A Standard Vocabulary represents a set of Vocabulary Elements whose definition and meaning must be agreed to in advance by trading partners who will exchange events using the vocabulary. For example, the EPCIS Core Data Definition Layer defines a vocabulary called "business step," whose elements are identifiers denoting such things as "shipping," "receiving," and so on. One trading partner may generate an event having a business step of "shipping," and another partner receiving that event through a query can interpret it because of a prior agreement as to what "shipping" means.

  Standard Vocabulary elements are defined in the CBV, as well as in more application-specific GS1 application standards, having been developed and vetted by user-driven GS1 workgroups. The master data associated with Standard Vocabulary elements are defined by those same organisations and tend to be distributed to users as part of a specification or by some similar means. New vocabulary elements within a given Standard Vocabulary are introduced through a very deliberate process, such as the ratification of a new version of a standard or through a vote of a GS1 workgroup. While an individual end user organisation acting alone may introduce a new Standard Vocabulary element, such an element would have limited use in a data exchange setting and would probably only be used within an organisation's four walls.

- *User Vocabulary:* A User Vocabulary represents a set of Vocabulary Elements whose definition and meaning are under the control of a single organisation or consortium. For example, the EPCIS Core Data Definition Layer defines a vocabulary called "business location," whose elements are identifiers denoting such things as "Acme Corp. Distribution Centre #3." Acme Corp may generate an event having a business location of "Acme Corp. Distribution Centre #3," and another partner receiving that event through a query can interpret it either because it correlates it with other events naming the same location, or by looking at master data attributes associated with the location, or both.

User Vocabulary elements are primarily defined by users or consortia. New vocabulary elements within a given User Vocabulary are introduced at the sole discretion of these parties, and trading partners must be prepared to respond accordingly. Usually, however, the rules for constructing new User Vocabulary Elements are established by organisations of multiple end users, and in any case must follow the rules defined in section 6.4 below.

The lines between these two kinds of vocabularies are subjective. Because the mechanisms defined in the EPCIS specification make no distinction between the two vocabulary types, it is never necessary to identify a particular vocabulary as belonging to one type or the other. The terms "Standard Vocabulary" and "User Vocabulary" are introduced only because they are useful as a hint as to the way a given vocabulary is expected to be defined and extended.

The CBV provides standardised vocabulary elements for many of the vocabulary types used in EPCIS event types. In particular, the CBV defines vocabulary elements for the following EPCIS Standard Vocabulary types: Business Step, Disposition, Persistent Disposition, Business Transaction Type, Source/Destination Type, Error Reason, and Sensor Property Type. The CBV also defines templates for constructing vocabulary elements for the following EPCIS User Vocabulary types: Object (EPC), Object Class (EPCClass), Location (Read Point and Business Location), Business Transaction ID, Source/Destination ID, Transformation ID, Event ID, Chemical Substance ID, Microorganism ID, and Resource ID.

6.3 Extension mechanisms

A key feature of EPCIS is its ability to be extended by different organisations to adapt to particular business situations. In all, the Abstract Data Model Layer provides five methods by which the data processed by EPCIS may be extended (the Service Layer, in addition, provides mechanisms for adding additional services), enumerated here from the most invasive type of extension to the least invasive:

- *New Event Type:* A new Event Type may be added in the Data Definition Layer. Adding a new Event Type requires each of the Data Definition Bindings to be extended, and may also require extension to the Capture and Query Interfaces and their Bindings.
- *New Event Field:* A new field may be added to an existing Event Type in the Data Definition Layer. The bindings, capture interface, and query interfaces defined in this specification are designed to permit this type of extension without requiring changes to the specification itself. (The same may not be true of other bindings or query languages defined outside this specification.)
- *New Vocabulary Type:* A new Vocabulary Type may be added to the repertoire of available Vocabulary Types. No change to bindings or interfaces are required.
- *New master data attribute:* A new attribute name may be defined for an existing Vocabulary. No change to bindings or interfaces are required.
- *New Instance/Lot master data (ILMD) Attribute:* A new attribute name may be defined for use in Instance/Lot master data (ILMD); see section 7.3.8. No change to bindings or interfaces are required.
- *New Vocabulary Element* A new element may be added to an existing Vocabulary.

The Abstract Data Model Layer has been designed so that most extensions arising from adoption by different industries or increased understanding within a given industry can be accommodated by the latter methods in the above list, which do not require revision to the specification itself. The more invasive methods at the head of the list are available, however, in case a situation arises that cannot be accommodated by the latter methods.

It is expected that there will be several different ways to extend the EPCIS specification, as summarised below:

Table 6-3 Ways to extend the EPCIS specification

How extension is disseminated	Responsible organisation	Extension method				
		New Event Type	New Event Field	New Vocabulary Type	New master data or ILMD (section 7.3.8) Attribute	New Vocabulary Element
New Version of EPCIS standard	GS1 EPCIS/CBV MSWG	Yes	Yes	Yes	Occasionally	Rarely
New Version of CBV standard	GS1 EPCIS/CBV MSWG	No	No	No	Yes	Yes (Standard Vocabulary, User Vocabulary template)
GS1 Application Standard for a specific industry	GS1 Application Standard Working Group for a specific industry	Rarely	Rarely	Occasionally	Yes	Yes (Standard Vocabulary)
GS1 Member Organisation Local Recommendation Document for a specific industry within a specific geography	GS1 Member Organisation	Rarely	Rarely	Occasionally	Yes	Yes (Standard Vocabulary)
Private Group Interoperability Specification	Industry Consortium or Private End User Group outside GS1	Rarely	Rarely	Occasionally	Yes	Yes (Standard Vocabulary)
Updated master data via EPCIS Query or other data sync	Individual End User	Rarely	Rarely	Rarely	Rarely	Yes (User vocabulary)

6.4 Identifier representation

The Abstract Data Model Layer introduces several kinds of identifiers, including Event Type names, Event Field names, Vocabulary names, Vocabulary Elements, and master data Attribute Names. Because all of these namespaces are open to extension, this specification imposes some rules on the construction of these names so that independent organisations may create extensions without fear of name collision.

Vocabulary Elements are subject to the following rules. In all cases, a Vocabulary Element is represented as Uniform Resource Identifier (URI) whose general syntax is defined in [RFC3986]. The types of URIs admissible as Vocabulary Elements are those URIs for which there is an owning authority. This includes:

- URI representations for EPC codes [TDS]. The owning authority for a particular EPC URI is the organisation to whom the GS1 Company Prefix (or other issuing authority, depending on the EPC scheme) was assigned.
- Absolute Uniform Resource Locators (URLs) [RFC1738]. The owning authority for a particular URL is the organisation that owns the Internet domain name in the authority portion of the URL.

- Uniform Resource Names (URNs) [RFC8141] in the `oid` namespace that begin with a Private Enterprise Number (PEN). The owning authority for an OID-URN is the organisation to which the PEN was issued.
- Uniform Resource Names (URNs) [RFC8141] in the `epc` or `epcglobal` namespace, other than URIs used to represent EPCs [TDS]. The owning authority for these URNs is GS1.
- GS1 Digital Link URIs in the form normatively specified in the GS1 Digital Link standard [GS1DL], restricted to a highly constrained set of GS1 Digital Link URIs that corresponds to each of the EPC Pure Identity URI schemes defined in TDS. (See section 8 of CBV for details.) The canonical form of GS1 Digital Link URIs is recommended but optional.

Event Type names and Event Field names are represented as namespace-qualified names (qnames), consisting of a namespace URI and a name. This has a straightforward representation in XML bindings that is convenient for extension.

6.5 Hierarchical vocabularies

Some Vocabularies have a hierarchical or multi-hierarchical structure. For example, a vocabulary of location names may have an element that means "Acme Corp. Retail Store #3" as well others that mean "Acme Corp. Retail Store #3 Backroom" and "Acme Corp. Retail Store #3 Sales Floor." In this example, there is a natural hierarchical relationship in which the first identifier is the parent and the latter two identifiers are children.

Hierarchical relationships between vocabulary elements are represented through master data. Specifically, a parent identifier carries, in addition to its master data attributes, a list of its children identifiers. Each child identifier SHALL belong to the same Vocabulary as the parent. In the example above, the element meaning "Acme Corp. Distribution Centre #3" would have a children list including the element that means "Acme Corp. Distribution Centre #3 Door #5."

Elsewhere in this specification, the term "direct or indirect descendant" is used to refer to the set of vocabulary elements including the children of a given vocabulary element, the children of those children, etc. In other words, the "direct or indirect descendants" of a vocabulary element include both the vocabulary elements from the next lower hierarchical level, as well as vocabulary elements in subsequent lower hierarchical levels.

A given element MAY be the child of more than one parent. This allows for more than one way of grouping vocabulary elements; for example, locations could be grouped both by geography and by function. An element SHALL NOT, however, be a child of itself, either directly or indirectly.

Non-Normative: Explanation: In the present version of this specification, only one hierarchical relationship is provided for, namely the relationship encoded in the special "children" list. Future versions of this specification may generalise this to allow more than one relationship, perhaps encoding each relationship via a different master data attribute.

Hierarchical relationships are given special treatment in queries (section 8.2), and may play a role in carrying out authorisation policies (section 8.2.2), but do not otherwise add any additional complexity or mechanism to the Abstract Data Model Layer.

The *GS1 GLN Allocation Rules Standard* [GLNAR3.0] provide a normative framework for identification of parties and locations.

7 Data definition layer

This section includes normative specifications of modules in the Data Definition Layer.

7.1 General rules for specifying data definition layer modules

The general rules for specifying modules in the Data Definition Layer are given here. These rules are then applied in section 7.2 to define the Core Event Types Module. These rules can also be applied by organisations wishing to layer a specification on top of this specification.

7.1.1 Content

In general, a Data Definition Module specification has these components, which populate the Abstract Data Model framework specified in section 6:

- *Value Types:* Definitions of data types that are used to describe the values of Event Fields and of master data attributes. The Core Event Types Module defines the primitive types that are available for use by all Data Definition Modules. Each Vocabulary that is defined is also implicitly a Value Type.
- *Event Types:* Definitions of Event Types, each definition giving the name of the Event Type (which must be unique across all Event Types) and a list of standard Event Fields for that type. An Event Type may be defined as a subclass of an existing Event Type, meaning that the new Event Type includes all Event Fields of the existing Event Type plus any additional Event Fields provided as part of its specification.
- *Event Fields:* Definitions of Event Fields within Event Types. Each Event Field definition specifies a name for the field (which must be unique across all fields of the enclosing Event Type) and the data type for values in that field. Event Field definitions within a Data Definition Module may be part of new Event Types introduced by that Module, or may extend Event Types defined in other Modules.
- *Vocabulary Types:* Definitions of Vocabulary Types, each definition giving the name of the Vocabulary (which must be unique across all Vocabularies), a list of standard master data attributes for elements of that Vocabulary, and rules for constructing new Vocabulary Elements for that Vocabulary. (Any rules specified for constructing Vocabulary Elements in a Vocabulary Type must be consistent with the general rules given in section 6.4)
- *Master data attributes:* Definitions of master data attributes for Vocabulary Types. Each master data attribute definition specifies a name for the Attribute (which must be unique across all attributes of the enclosing Vocabulary Type) and the data type for values of that attribute. Master data definitions within a Data Definition Module may belong to new Vocabulary Types introduced by that Module, or may extend Vocabulary Types defined in other Modules.
- *Vocabulary Elements:* Definitions of Vocabulary Elements, each definition specifying a name (which must be unique across all elements within the Vocabulary, and conform to the general rules for Vocabulary Elements given in section 6.4 as well as any specific rules specified in the definition of the Vocabulary Type), and optionally specifying master data (specific attribute values) for that element.

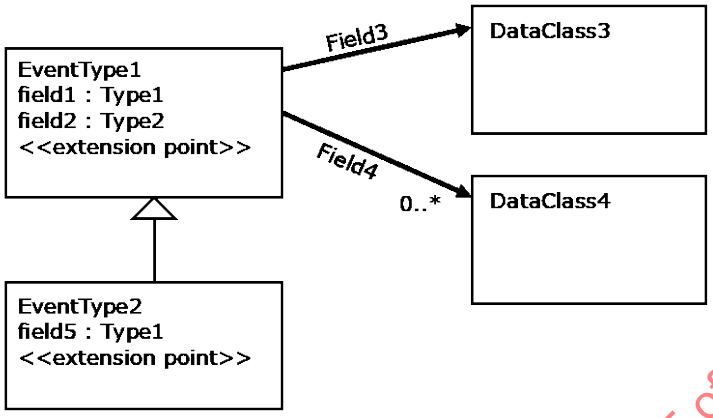
Non-Normative: Amplification: As explained in section 6.3, Data Definition Modules defined in this specification and by companion specifications developed by the EPCIS Working Group will tend to include definitions of Value Types, Event Types, Event Fields, and Vocabulary Types, while modules defined by other groups will tend to include definitions of Event Fields that extend existing Event Types, master data attributes that extend existing Vocabulary Types, and Vocabulary Elements that populate existing Vocabularies. Other groups may also occasionally define Vocabulary Types.

The word "Vocabulary" is used informally to refer to a Vocabulary Type and the set of all Vocabulary Elements that populate it.

7.1.2 Notation

In the sections below, Event Type and Event fields are specified using a restricted form of UML class diagram notation. UML class diagrams used for this purpose may contain classes that have attributes (fields) and associations, but not operations.

Figure 7-1 EPCIS data definition notation



The example UML diagram above shows a data definition for two Event Types, `EventType1` and `EventType2`. These event types make use of four Value Types: `Type1`, `Type2`, `DataClass3`, and `DataClass4`. `Type1` and `Type2` are primitive types, while `DataClass3` and `DataClass4` are complex types whose structure is also specified in UML.

The Event Type `1` in this example has four fields. `Field1` and `Field2` are of primitive type `Type1` and `Type2` respectively. `1` has another field `Field3` whose type is `DataClass3`. Finally, `1` has another field `Field4` that contains a list of zero or more instances of type `DataClass4` (the "0..*" notation indicates "zero or more").

This diagram also shows a data definition for `EventType2`. The arrow with the open-triangle arrowhead indicates that `EventType2` is a subclass of `EventType1`. This means that `EventType2` actually has five fields: four fields inherited from `EventType1` plus a fifth `field5` of type `Type1`.

Within the UML descriptions, the notation `<<extension point>>` identifies a place where implementations SHALL provide for extensibility through the addition of new data members. (When one type has an extension point, and another type is defined as a subclass of the first type and also has an extension point, it does not mean the second type has two extension points; rather, it merely emphasises that the second type is also open to extension.) Extensibility mechanisms SHALL provide for both proprietary extensions by vendors of EPCIS-compliant products, and for extensions defined by GS1 through future versions of this specification or through new specifications.

In the case of the standard XML bindings, the extension points are implemented within the XML schema following the methodology described in section 9.1.

All definitions of Event Types SHALL include an extension point, to provide for the extensibility defined in section 6.3 ("New Event Fields"). Value Types MAY include an extension point.

7.1.3 Semantics

Each event (an instance of an Event Type) encodes several assertions which collectively define the semantics of the event. Some of these assertions say what was true at the time the event was captured. Other assertions say what is expected to be true following the event, until invalidated by a subsequent event. These are called, respectively, the *retrospective semantics* and the *prospective semantics* of the event.

For example, if widget #23 enters building #5 through door #6 at 11:23pm, then one retrospective assertion is that "widget #23 was observed at door #6 at 11:23pm,", while a prospective assertion is that "widget #23 is in building #5." The key difference is that the retrospective assertion refers to a specific time in the past ("widget #23 *was observed*…"), while the prospective assertion is a statement about the present condition of the object ("widget #23 *is in*…"). The prospective assertion presumes that if widget #23 ever leaves building #5, another EPCIS capture event will be recorded to supersede the prior one.

In general, retrospective semantics are things that were incontrovertibly known to be true at the time of event capture, and can usually be relied upon by EPCIS Accessing Applications as accurate statements of historical fact. Prospective semantics, since they attempt to say what is true after an event has taken place, must be considered at best to be statements of "what ought to be" rather than of "what is." A prospective assertion may turn out not to be true if the capturing apparatus does not function perfectly, or if the business process or system architecture were not designed to capture EPCIS events in all circumstances. Moreover, in order to make use of a prospective assertion implicit in an event, an EPCIS Accessing Application must be sure that it has access to any subsequent event that might supersede the event in question.

The retrospective/prospective dichotomy plays an important role in EPCIS's definition of location, in section 7.3.4.

In certain situations, an earlier event is subsequently discovered to be in error (the assertions its semantics makes are discovered to be incorrect), and the error cannot be corrected by recording a new event that adds additional assertions through its own semantics. For these cases, a mechanism is provided to record an event whose semantics assert that the assertions previously made by the erroneous event are in error. See section 7.4.1.2.

7.2 Core event types module – overview

The Core Event Types data definition module specifies the Event Types that represent EPCIS data capture events. These events are typically generated by an EPCIS Capturing Application and provided to EPCIS infrastructure using the data capture operations defined in section 8.1. These events are also returned in response to query operations that retrieve events according to query criteria.

The components of this module, following the outline given in section 7.1.1, are as follows:

- *Value Types:* Primitive types defined in section 7.3.1 and 7.3.2.
- *Event Types:* Event types as shown in the UML diagram in section 7.2.1, and defined in section 7.4.
- *Event Fields:* Included as part of the Event Types defined in section 7.4.
- *Vocabulary Types:* Types defined in section 7.3.3 through 7.3.8, and summarised in section 7.1.1.
- *Master data attributes:* Included as part of Vocabulary Types definitions. It is expected that industry vertical working groups will define additional master data attributes for the vocabularies defined here.
- *Vocabulary Elements:* None provided as part of this specification. It is expected that industry vertical working groups will define vocabulary elements for the `BusinessStep` vocabulary (section 7.3.6), the `Disposition` vocabulary (section 7.3.6.2), and the `BusinessTransactionType` vocabulary (section 7.3.6.3.1).

This module defines six event types, one very generic event and five subclasses that can represent events arising from supply chain activity across a wide variety of industries:

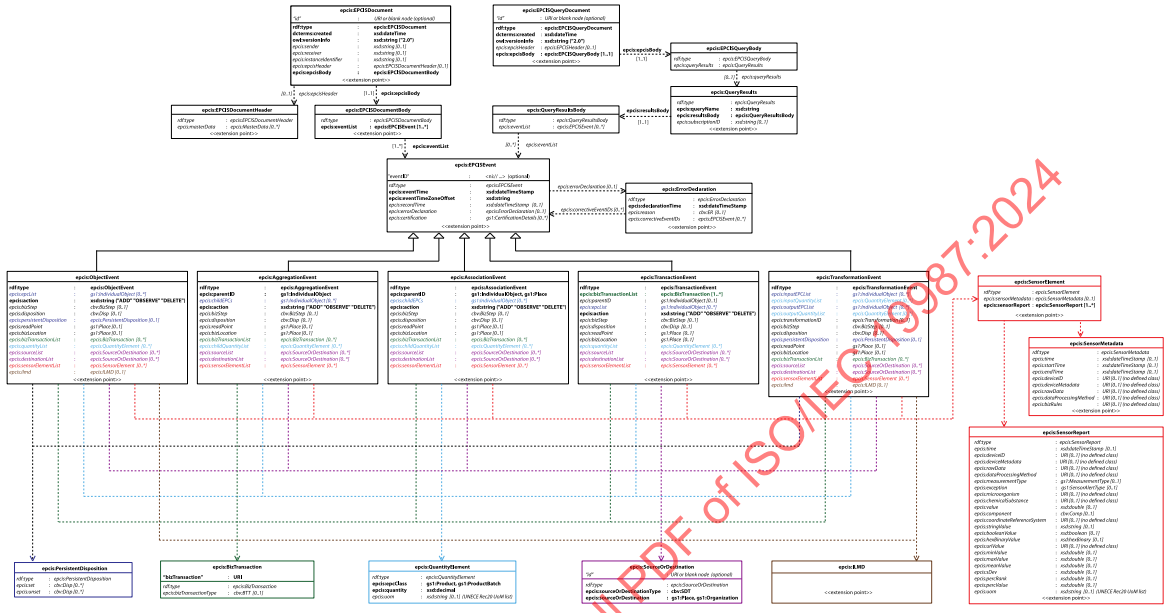
- `EPCISEvent` (section 7.4.1) is a generic base class for all event types in this module as well as others.
- `ObjectEvent` (section7.4.2) represents an event that happened to one or more physical or digital objects.
- `AggregationEvent` (section 7.4.3) represents an event that happened to one or more objects that are physically aggregated together (physically constrained to be in the same place at the same time, as when cases are aggregated to a pallet).
- `TransactionEvent` (section 7.4.4) represents an event in which one or more objects become associated or disassociated with one or more identified business transactions.
- `TransformationEvent` (section 7.4.5) represents an event in which input objects are fully or partially consumed and output objects are produced, such that any of the input objects may have contributed to all of the output objects.
- `AssociationEvent` (section 7.4.6) is similar to an AggregationEvent, but allows for associations of objects with physical locations, and is especially suited to capture parent-child relationships that persist even after more temporarily linked children are disassociated from the parent.

A UML diagram showing these Event Types is depicted in the following section.

7.2.1 UML Diagrams of EPCIS Event Types

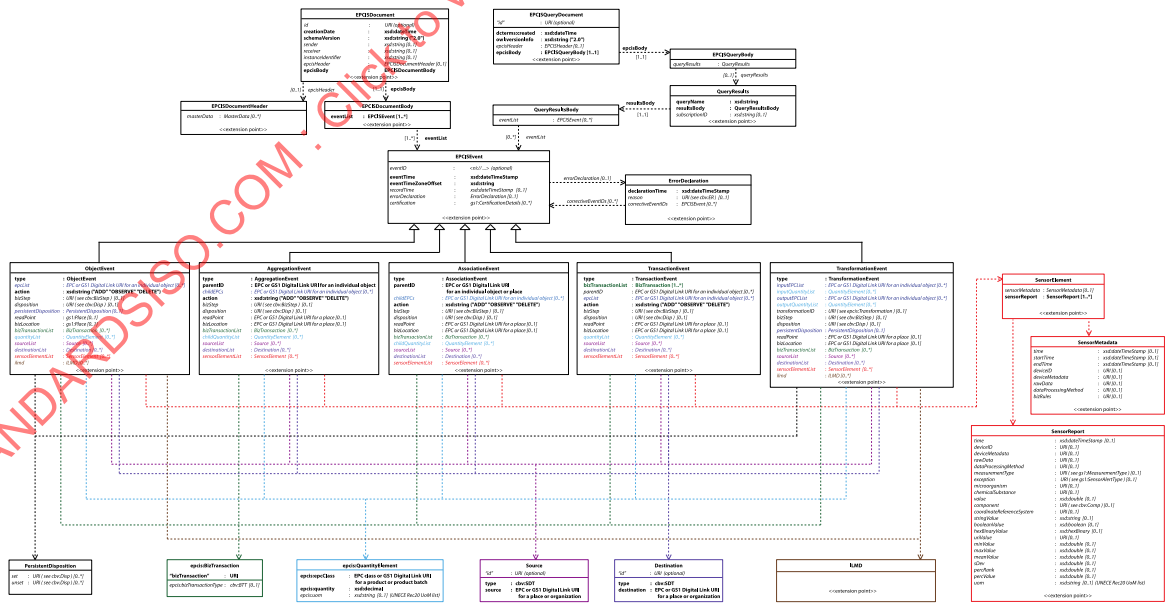
7.2.1.1 UML with Ontology focus

Figure 7-2 EPCIS UML with Ontology focus



7.2.1.2 UML with Syntax focus

Figure 7-3 EPCIS UML with Syntax focus



7.2.1.3 UML for SBDH

Note: The UML class diagrams for SBDH, which are not central to the EPCIS event data model, are included in section 15.

7.2.2 Overview of EPCIS event "dimensions" (non-normative)

Each of the core event types (not counting the generic `EPCISEvent`) has fields that represent five key dimensions of any EPCIS event. These five dimensions are: (1) the object(s) or other entities that are the subject of the event; (2) the date and time; (3) the location at which the event occurred and/or the whereabouts of the objects subsequent to the event; (4) the business context; (5) the condition of the objects that are the subject of the event. These five dimensions may be conveniently remembered as "**what**, **when**, **where**, **why** and **how**" (respectively). The "what" dimension varies depending on the event type (e.g., for an `ObjectEvent` the "what" dimension is one or more EPCs; for an `AggregationEvent` the "what" dimension is a parent ID and list of child EPCs). The "where" and "why" dimensions have both a retrospective aspect and a prospective aspect (see section 7.1.3), represented by different fields. The "how" dimension is populated on the basis of captured sensor data.

The following table summarises the fields of the event types that pertain to the five key dimensions;

in addition to the fields belonging to the five key dimensions, events may carry additional descriptive information in other fields, also indicated in the table below.

Dimension in EPCIS/CBV 1.x	Categorisation in EPCIS/CBV 2.0		Field	EPCIS section in which the field is defined	CBV section in which its value range is specified
WHAT	Objects in Focus (WHAT) 	Instance	epcList	7.4.2 ObjectEvent 7.4.4 TransactionEvent	*EPC Tag Data Standard (TDS) section 6,"EPC URI"* 8.2 Physical or Digital Objects (Instance)
			parentID	7.4.3 AggregationEvent 7.4.4 TransactionEvent 7.4.6 AssociationEvent	
			childEPCs	7.4.3 AggregationEvent 7.4.6 AssociationEvent	
			inputEPCList outputEPCList	7.4.5 TransformationEvent	
		Class	quantityList	7.4.2 ObjectEvent 7.4.4 TransactionEvent	*EPC Tag Data Standard (TDS) section 8,"URIs for EPC Pure Identity Patterns"* 8.3 Physical or Digital Objects (Class)
			childQuantityList	7.4.3 AggregationEvent 7.4.6 AssociationEvent	
			inputQuantityList outputQuantityList	7.4.5 TransformationEvent	

Dimension in EPCIS/CBV 1.x	Categorisation in EPCIS/CBV 2.0	Field	EPCIS section in which the field is defined	CBV section in which its value range is specified
WHEN	Chronology (WHEN) 	eventTime	7.4.1 EPCISEvent	
		eventTimeZoneOffset		
		recordTime	7.4.1 EPCISEvent	
WHERE	Whereabouts (WHERE) 	readPoint	7.4.2 ObjectEvent 7.4.3 AggregationEvent 7.4.4 TransactionEvent 7.4.5 TransformationEvent 7.4.6 AssociationEvent	8.4 Locations
		bizLocation		
n/a	Condition (HOW) 	sensorElementList	7.4.2 ObjectEvent 7.4.3 AggregationEvent 7.4.4 TransactionEvent 7.4.5 TransformationEvent 7.4.6 AssociationEvent	7.6 Sensor Measurement Types 8.9 Chemical substance identifiers 8.10 Microorganism identifiers
WHY	Business Context (WHY) 	bizStep	7.4.2 ObjectEvent 7.4.3 AggregationEvent 7.4.4 TransactionEvent 7.4.5 TransformationEvent	7.1 Business Steps
		bizTransactionList		8.5 Business Transactions
		disposition		7.1 Dispositions

Dimension in EPCIS/CBV 1.x	Categorisation in EPCIS/CBV 2.0	Field	EPCIS section in which the field is defined	CBV section in which its value range is specified
		persistentDisposition	7.4.6 AssociationEvent	7.1 Dispositions
		sourceList		8.6 Source/Destination Identifiers
		destinationList		
	Other fields	ilmd	7.3.7 Instance/Lot master data (ILMD)	9 Trade Item Master Data
(core field)		action	7.3.2 Action type	
(transformationID)		transformationID	7.4.5 TransformationEvent	8.7 Transformation Identifiers
(core field)		eventID	7.4.1 EPCISEvent	8.8 Event Identifiers
(core field)		errorDeclaration	7.4.1 EPCISEvent	7.5 Error Reason Identifiers 8.8 Event Identifiers

It is expected that the majority of additional descriptive information fields will be defined by industry-specific specifications layered on top of the core EPCIS and CBV standards.

7.2.3 Table of vocabulary types

The following table summarises the vocabulary types defined in this module. The URI column gives the formal name for the vocabulary used when the vocabulary must be referred to by name across the EPCIS interface.

Vocabulary type	Section	User or standard vocabulary	URI
ReadPointID	7.3.5	User	urn:epcglobal:epcis:vtype:ReadPoint
BusinessLocationID	7.3.5	User	urn:epcglobal:epcis:vtype:BusinessLocation
BusinessStepID	7.3.6.1	Standard	urn:epcglobal:epcis:vtype:BusinessStep
DispositionID	7.3.6.2	Standard	urn:epcglobal:epcis:vtype:Disposition
BusinessTransaction	7.3.6.3	User	urn:epcglobal:epcis:vtype:BusinessTransaction
BusinessTrasactionTypeID	7.3.6.3.1	Standard	urn:epcglobal:epcis:vtype:BusinessTransactionType
EPCClass	7.3.3.1.2	User	urn:epcglobal:epcis:vtype:EPCClass
SourceDestTypeID	7.3.6.4.1	Standard	urn:epcglobal:epcis:vtype:SourceDestType
SourceDestID	7.3.6.4.2	User	urn:epcglobal:epcis:vtype:SourceDest
LocationID	7.3.5	User	urn:epcglobal:epcis:vtype:Location
PartyID	7.3.6.4.2	User	urn:epcglobal:epcis:vtype:Party
ErrorReasonID	7.4.1.2	Standard	urn:epcglobal:epcis:vtype:ErrorReason
SensorPropertyTypeID	7.3.7.1.4	Standard	urn:epcglobal:epcis:vtype:SensorPropertyType
MicroorganismID	7.3.7.1.6	User	urn:epcglobal:epcis:vtype:Microorganism
ChemicalSubstanceID	7.3.7.1.7	User	urn:epcglobal:epcis:vtype:ChemicalSubstance
ResourceID	7.3.7.1.8	User	urn:epcglobal:epcis:vtype:Resource

7.3 Core event types module – building blocks

This section specifies the building blocks for the event types defined in section 7.4.

7.3.1 Primitive types

The following primitive types are used within the Core Event Types Module.

Type	Description
xsd:int	An integer. Range restrictions are noted where applicable.

Type	Description
xsd:dateTimeStamp	A timestamp, giving the date and time in a time zone-independent manner. For bindings in which fields of this type are represented textually, an ISO 8601 [ISO8601] compliant representation SHOULD be used.
EPC	An Electronic Product Code Pure Identity URI, as defined in [TDS], or a GS1 Digital Link URI, as provided for in CBV section 8.
xsd:float	Patterned after the IEEE single-precision 32-bit floating point datatype; often used to approximate arbitrary real numbers. The ·value space· of float contains the non-zero numbers $m \times 2e$, where m is an integer whose absolute value is less than 224, and e is an integer between −149 and 104, inclusive.
xsd:double	The double-precision floating point datatype is patterned after the IEEE double-precision 64-bit floating point datatype, specified in IEEE 754-2008 [IEEE754]. Each double-precision floating point datatype has a value space that is a subset of the rational numbers. Floating point numbers are often used to approximate arbitrary real numbers.
xsd:boolean	Represents the values of two-valued logic; has the ·value space· of two-valued logic: {true, false}.
xsd:hexBinary	Represents arbitrary hex-encoded binary data; the ·value space· of hexBinary is the set of finite-length sequences of zero or more binary octets.
xsd:anyURI	Represents an Internationalized Resource Identifier Reference (IRI); an anyURI value can be absolute or relative.
xsd:decimal	Represents a subset of the real numbers, which can be represented by decimal numerals. The ·value space· of decimal is the set of numbers that can be obtained by dividing an integer by a non-negative power of ten, i.e., expressible as i / 10n where i and n are integers and n ≥ 0. Precision is not reflected in this value space; the number 2.0 is not distinct from the number 2.00.
xsd:string	Represents character strings.

The EPC type is defined as a primitive type for use in events when referring to EPCs that are not part of a Vocabulary Type. For example, an SGTIN EPC used to denote an instance of a trade item in the epcList field of an ObjectEvent is an instance of the EPC primitive type. But an SGLN EPC used as a read point identifier (section 7.3.4) in the ReadPoint field of an ObjectEvent is a Vocabulary Element, not an instance of the EPC primitive type.

Non-Normative: Explanation: This reflects a design decision not to consider individual trade item instances as Vocabulary Elements having master data, because trade item instances are constantly being created and hence new EPCs representing trade items are constantly being commissioned. In part, this design decision reflects consistent treatment of master data as excluding data that grows as more business is transacted (see comment in section 6.1), and in part reflects the pragmatic reality that data about trade item instances is likely to be managed more like event data than master data when it comes to aging, database design, etc.

7.3.2 Action type

The Action type says how an event relates to the lifecycle of the entity being described. For example, AggregationEvent (section 7.4.3) is used to capture events related to aggregations of objects, such as cases aggregated to a pallet. Throughout its life, the pallet load participates in many business process steps, each of which may generate an EPCIS event. The action field of each event says how the aggregation itself has changed during the event: have objects been added to the aggregation, have objects been removed from the aggregation, or has the aggregation simply been observed without change to its membership? The action is independent of the

bizStep (of type BusinessStepID) which identifies the specific business process step in which the action took place.

The Action type is an enumerated type having three possible values:

Action value	Meaning
ADD	The entity in question has been created or added to.
OBSERVE	The entity in question has not been changed: it has neither been created, added to, destroyed, or removed from.
DELETE	The entity in question has been removed from or destroyed altogether.

The description below for each event type that includes an Action value says more precisely what Action means in the context of that event.

Note that the values above are the only values possible for Action. Unlike other types defined below, Action is *not* a vocabulary type, and SHALL NOT be extended by industry groups.

7.3.3 The "What" dimension

Section 8.2 and section 8.3 of the CBV defines the data fields and the expected value types and data types used in the "What" dimension of EPCIS events, including discussion of class-level vs instance-level identification.

7.3.3.1 QuantityElement

A QuantityElement is a structure that identifies objects identified by a specific class-level identifier, either a specific quantity or an unspecified quantity. It has the following structure:

Field	Type	Description
epcClass	EPCClass	A class-level identifier for the class to which the specified quantity of objects belongs.
quantity	xsd:double	(Optional) A number that specifies how many or how much of the specified EPCClass is denoted by this QuantityElement. The quantity may be omitted to indicate that the quantity is unknown or not specified. If quantity is omitted, then uom SHALL be omitted as well. Otherwise, if quantity is specified: If the QuantityElement lacks a uom field (below), then the quantity SHALL have a positive integer value, and denotes a count of the number of instances of the specified EPCClass that are denoted by this QuantityElement. If the QuantityElement includes a uom, then the quantity SHALL have a positive value (but not necessarily an integer value), and denotes the magnitude of the physical measure that specifies how much of the specified EPCClass is denoted by this QuantityElement
uom	UOM	(Optional) If present, specifies a unit of measure by which the specified quantity is to be interpreted as a physical measure, specifying how much of the specified EPCClass is denoted by this QuantityElement. The uom SHALL be omitted if quantity is omitted.

EPCClass is a Vocabulary whose elements denote classes of objects. EPCClass is a User Vocabulary as defined in section 6.2. Any EPC whose structure incorporates the concept of object class can be referenced as an EPCClass.

An EPCClass may refer to a class having fixed measure or variable measure. A fixed measure class has instances that may be counted; for example, a GTIN that refers to fixed-size cartons of a product. A variable measure class has instances that cannot be counted and so the quantity is specified as a physical measure; for example, a GTIN that refers to copper wire that is sold by length, carpeting that is sold by area, bulk oil

that is sold by volume, or fresh produce that is sold by weight. The following table summarises how the `quantity` and `uom` fields are used in each case:

EPCClass	`quantity` field	`uom` field	Meaning
Fixed measure	Positive integer	Omitted	The `quantity` field specifies the count of the specified class.
Variable measure	Positive number, not necessarily an integer	Present	The `quantity` field specifies the magnitude, and the `uom` field the physical unit, of a physical measure describing the amount of the specified class.
Fixed or Variable Measure	Omitted	Omitted	The quantity is unknown or not specified.

Master data attributes for the `EPCClass` vocabulary contain whatever master data is defined for the referenced objects independent of EPCIS (for example, product catalogue data); definitions of these are outside the scope of this specification.

7.3.3.1.1 UOM

As specified above, the `uom` field of a `QuantityElement` is present when the `QuantityElement` uses a physical measure to specify the quantity of the specified `EPCClass`. When a `uom` field is present, its value SHALL be the 2- or 3-character code for a physical unit specified in the "Common Code" column of UN/CEFACT Recommendation 20 [CEFACT20]. Moreover, the code SHALL be a code contained in a row of [CEFACT20] meeting all of the following criteria:

- The "Quantity" column contains one of the following quantities: *length*, *area*, *volume*, or *mass*.
- The "Status" column does *not* contain "X" (deleted) or "D" (deprecated).

For purposes of the first criterion, the quantity must appear as a complete phrase. Example: "metre" (MTR) is allowed, because the quantity includes *length* (among other quantities such as *breadth*, *height*, etc.). But "pound-force per foot" (F17) is *not* allowed, because the quantity is *force divided by length*, not just *length*.

7.3.3.1.2 Class-level identifiers

Class-level identifiers are specified -- as EPC URNs and GS1 Digital Link URIs -- in section 8.3 of CBV.

Implementations SHALL understand queries expressed in both EPC URN and GS1 Digital Link URI syntaxes, and MAY return query responses corresponding to either syntax, at the discretion of the responding implementation. The requesting client might need to translate the identifiers within the query response into its preferred syntax.

7.3.3.2 Identifier types (Non-Normative)

The normative specifications of identifiers are in the EPC Tag Data Standard [TDS], the EPC Core Business Vocabulary [CBV] and the GS1 Digital Link Standard [DL].

7.3.4 The "When" dimension

The "When" dimension of EPCIS includes data fields that provide a chronological context to visibility events.

7.3.4.1 The "When" dimension in the EPCISEvent common base type

The EPCISEvent common base type includes the `eventTime`, `recordTime` and `eventTimeZoneOffset` fields, as specified in section 7.4.1.

7.3.4.2 The "When" dimension in the Error Declaration

The EPCIS Error Declaration element, specified in section 7.4.1.2.1 , includes -- beyond the reiterated fields of the original, erroneous event -- the `declarationTime` field.

7.3.4.3 The "When" dimension in Sensor Metadata

EPCIS Sensor Metadata, specified in section 7.3.7.1.1, includes the `time`, `startTime`, and `endTime` fields, which relate to the time of sensor data capture (i.e., rather than the time of the EPCIS *event* capture).

7.3.5 The "Where" Dimension – read point and business location

This section addresses the notion of location information as used in EPCIS.

EPCIS location types are defined as EPCIS Vocabulary Types, as follows:

Type	Description
`ReadPointID`	A Read Point is a discretely recorded location that is meant to identify the most specific place at which an EPCIS event took place. Read Points are determined by the EPCIS Capturing Application, perhaps inferred as a function of logical reader if stationary readers are used, perhaps determined directly by reading a location tag if the reader is mobile, or in general determined by any other means the EPCIS Capturing Application chooses to use. Conceptually, the Read Point is designed to identify "where objects were at the time of the EPCIS event."
`BusinessLocationID`	A Business Location is a uniquely identified and discretely recorded location that is meant to designate the specific place where an object is assumed to be following an EPCIS event until it is reported to be at a different Business Location by a subsequent EPCIS event. As with the Read Point, the EPCIS Capturing Application determines the Business Location based on whatever means it chooses. Conceptually, the Business Location is designed to identify "where objects are following the EPCIS event."

`ReadPointID` and `BusinessLocationID` are User Vocabularies as defined in section 6.2. Some industries may wish to use EPCs as vocabulary elements, in which case pure identity URIs as defined in [TDS] SHALL be used.

Note:

The `LocationID` type is a supertype of `ReadPointID`, `BusinessLocationID`, and `SourceDestID`. In an EPCIS master data document (or master data header within an EPCIS document or EPCIS query document), the `urn:epcglobal:epcis:vtype:Location` URI may be used to specify a single vocabulary containing identifiers that may appear in the read point, business step, source, or destination field of associated EPCIS events.

Non-Normative: Illustration: For example, in industries governed by GS1 General Specifications, `readPointID`, and `businessLocationID` may be SGLN-URIs.

Location vocabulary elements are not *required* to be EPCs.

Non-Normative: Explanation: Allowing non-EPC URIs for locations gives organisations greater freedom to reuse existing ways of naming locations.

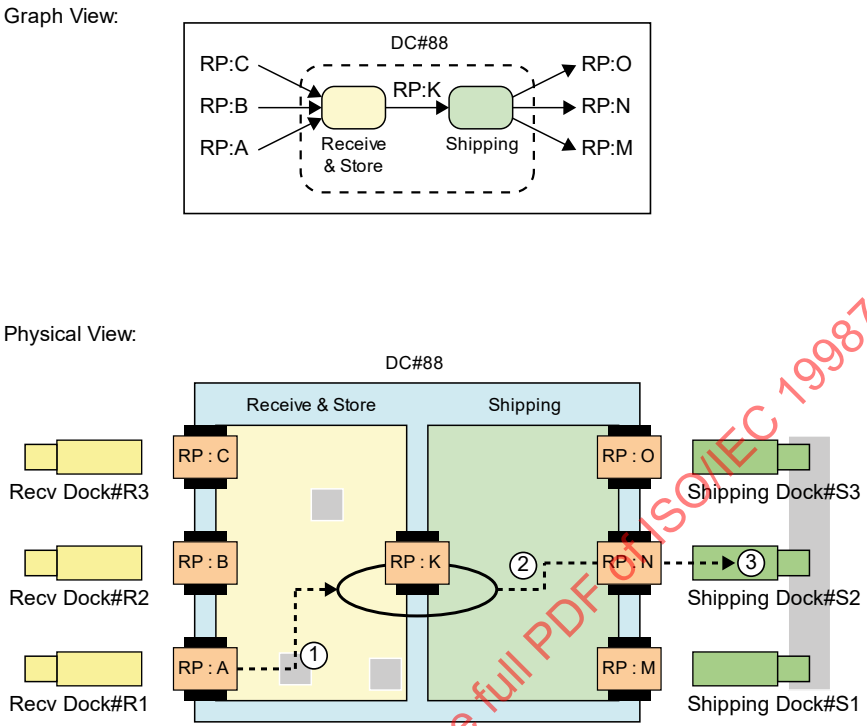
For all of the EPCIS Event Types defined in this section, capture events include separate fields for Read Point and Business Location. In most cases, both are optional, so that it is still possible for an EPCIS Capturing Application to include partial information if both are not known.

Non-Normative: Explanation: Logical Reader and Physical Reader are omitted from the definitions of EPCIS events in this specification. Physical Reader is generally not useful information for exchange between partners. For example, if a reader malfunctions and is replaced by another reader of identical make and model, the Physical Reader ID has changed. This information is of little interest to trading partners. Likewise, the Logical Reader ID may change if the capturing organisation makes a change in the way a particular business process is executed; again, not often of interest to a partner.

The distinction between Read Point and Business Location is very much related to the dichotomy between retrospective semantics and prospective semantics discussed above. In general, Read Points play a role in retrospective semantics, while Business Locations are involved in prospective statements. This is made explicit in the way each type of location enters the semantic descriptions given at the end of each section below that defines an EPCIS capture event.

7.3.5.1 Example of the distinction between a read point and a business location (Non-Normative)

Figure 7-4 Example of the distinction between a read point and a business location



Tag	Time	Read Point	Business Location	Comment
#123	07:00	"RP-DC#88-A"	DC#88.Receive & Store	Product entered DC via DockDoor#R1
#123	09:00	"RP-DC#88-K"	DC#88.Shipping	Product placed on conveyor for shipping
#123	09:30	"RP-DC#88-N"	[omitted]	Product shipped via dock door#S2

The figure above shows a typical use case consisting of rooms with fixed doorways at the boundaries of the rooms. In such a case, Read Points correspond to the doorways (with RFID instrumentation) and Business Locations correspond to the rooms. Note that the Read Points and Business Locations are not in one-to-one correspondence; the only situation where Read Points and Business Locations could have a 1:1 relationship is the unusual case of a room with a single door, such a small storeroom.

Still considering the rooms-and-doors example, the Business Location is usually the location type of most interest to a business application, as it says which room an object is in. Thus, it is meaningful to ask the inventory of a Business Location such as the backroom. In contrast, the Read Point indicates the doorway through which the object entered the room. It is not meaningful to ask the inventory of a doorway. While sometimes not as relevant to a business application, the Read Point is nevertheless of significant interest to higher level software to understand the business process and the final status of the object, particularly in the presence of less than 100% read rates. Note that correct designation of the business location requires both that the tagged object be observed at the Read Point and that the direction of movement be correctly determined – again reporting the Read Point in the event will be very valuable for higher level software.

A supply chain like the rooms-and-doors example may be represented by a graph in which each node in the graph represents a room in which objects may be found, and each arc represents a doorway that connects two rooms. Business Locations, therefore, correspond to nodes of this graph, and Read Points correspond to the arcs. If the graph were a straight, unidirectional chain, the arcs traversed by a given object could be reconstructed from knowing the nodes; that is, Read Point information would

be redundant given the Business Location information. In more real-world situations, however, objects can take multiple paths and move "backwards" in the supply chain. In these real-world situations, providing Read Point information in addition to Business Location information is valuable for higher level software.

The key to balancing seemingly conflicting requirements is to define and relate various location types, and then to rely on the EPCIS Capturing Application to properly record them for a given capture event. This is why EPCIS events contain both a ReadPointID and a BusinessLocationID (the two primitive location types).

7.3.6 The "Why" dimension

This section defines the data fields and the expected value types and data types used in the "Why" dimension of the event types specified in section 7.3.6.4.

7.3.6.1 Business step

`BusinessStepID` is a vocabulary whose elements denote steps in business processes. An example is an identifier that denotes "shipping." The business step field of an event specifies the business context of an event: what business process step was taking place that caused the event to be captured? `BusinessStepID` is an example of a Standard Vocabulary as defined in section 6.2.

Non-Normative: Explanation: Using an extensible vocabulary for business step identifiers allows GS1 standards (including and especially the GS1 Core Business Vocabulary) to define some common terms such as "shipping" or "receiving," while allowing for industry groups and individual end-users to define their own terms. Master data provides additional information.

This specification defines no master data attributes for business step identifiers.

7.3.6.2 Disposition and Persistent Disposition

`DispositionID` is a vocabulary whose elements denote a business state of an object. An example is an identifier that denotes "recalled."

7.3.6.2.1 Disposition

The disposition field of an event specifies the business condition of the event's objects, subsequent to the event. The disposition is assumed to hold true until another event indicates a change of disposition. Intervening events that do not specify a disposition field have no effect on the presumed disposition of the object. `DispositionID` is an example of a Standard Vocabulary as defined in section 6.2.

Applicability of `Disposition` by event type

event type	`Action` value	`Disposition` applies to

`ObjectEvent`	`ADD`
	`OBSERVE`
	`DELETE`
		`epcList`
		`quantityList`

`AggregationEvent`	`ADD`
	`OBSERVE`
		`parentID`
		`childEPCs`
		`childQuantityList`

	`DELETE`	`childEPCs` `childQuantityList`

`TransactionEvent`	`ADD`
	`OBSERVE`
	`DELETE`
		`parentID`
		`epcList`
		`quantityList`

`TransformationEvent`	n/a	`outputEPCList` `outputQuantityList`

`AssociationEvent`	`ADD`
	`OBSERVE`
		`parentID`
		`childEPCs`
		`childQuantityList`

	`DELETE`	`childEPCs` `childQuantityList`

7.3.6.2.2 Persistent Disposition

The `persistentDisposition` field of an event specifies one or more business conditions of the event's objects, subsequent to the event. One or more `persistentDisposition` values can be `set` or `unset`, independently of each other. Values that are `set` are considered to remain valid until explicitly `unset`. Like disposition, persistentDisposition leverages the `DispositionID` Standard Vocabulary.

NOTE: `persistentDisposition` SHOULD only be used in conjunction with `ObjectEvent`s with `Action` type `OBSERVE`, in order to avoid the potential for ambiguity with regard to inheritance of `persistentDisposition` by child/input/output objects.

Non-Normative: Explanation: Using an extensible vocabulary for disposition identifiers allows GS1 standards (including and especially the GS1 Core Business Vocabulary) to define some common terms such as "recalled" or "in transit," while allowing for industry groups and individual end-users to define their own terms. Master data may provide additional information.

This specification defines no master data attributes for disposition identifiers.

Applicability of `persistentDisposition` by event type

event type	`Action` value	`set` or `unset` of `persistentDisposition` applies to

`ObjectEvent`	`ADD`
	`OBSERVE`
	`DELETE`
		`epcList`
		`quantityList`

`AggregationEvent`	`ADD`
	`OBSERVE`
	`DELETE`
		n/a

`TransactionEvent`	`ADD`
	`OBSERVE`
	`DELETE`
		n/a

`TransformationEvent`	n/a	`outputEPCList` `outputQuantityList`

`AssociationEvent`	`ADD`
	`OBSERVE`
	`DELETE`
		n/a

7.3.6.3 Business transaction

A `BusinessTransaction` identifies a particular business transaction. An example of a business transaction is a specific purchase order. Business Transaction information may be included in EPCIS events to record an event's participation in particular business transactions.

A business transaction is described in EPCIS by a structured type consisting of a pair of identifiers, as follows.

Field	Type	Description
`type`	`BusinessTransactionTypeID`	(Optional) An identifier that indicates what kind of business transaction this `BusinessTransaction` denotes. If omitted, no information is available about the type of business transaction apart from what is implied by the value of the `bizTransaction` field itself.
`bizTransaction`	`BusinessTransactionID`	An identifier that denotes a specific business transaction.

The two vocabulary types `BusinessTransactionTypeID` and `BusinessTransactionID` are defined in the sections below.

7.3.6.3.1 Business transaction type

`BusinessTransactionTypeID` is a vocabulary whose elements denote a specific type of business transaction. An example is an identifier that denotes "purchase order." `BusinessTransactionTypeID` is an example of a Standard Vocabulary as defined in section 6.2.

Non-Normative: Explanation: Using an extensible vocabulary for business transaction type identifiers allows GS1 standards to define some common terms such as "purchase order" while allowing for industry groups and individual end-users to define their own terms. Master data may provide additional information.

This specification defines no master data attributes for business transaction type identifiers.

7.3.6.3.2 Business transaction ID

`BusinessTransactionID` is a vocabulary whose elements denote specific business transactions. An example is an identifier that denotes "Acme Corp purchase order number 12345678." `BusinessTransactionID` is a User Vocabulary as defined in section 6.2.

Non-Normative: Explanation: URIs are used to provide extensibility and a convenient way for organisations to distinguish one kind of transaction identifier from another. For example, if Acme Corporation has purchase orders (one kind of business transaction) identified with an 8-digit number as well as shipments (another kind of business transaction) identified by a 6-character string, the following business transaction IDs might be associated with a particular EPC over time:

https://transaction.acme.com/po/12345678
https://transaction.acme.com/shipment/34ABC8

An EPCIS Accessing Application might query EPCIS and discover both of the transaction IDs; using URIs gives the application a way to understand which ID is of interest to it.

7.3.6.4 Source and destination

A `Source` or `Destination` is used to provide additional business context when an EPCIS event is part of a business transfer; that is, a process in which there is a transfer of ownership, responsibility, and/or custody of physical or digital objects.

In many cases, a business transfer requires several individual business steps (and therefore several EPCIS events) to execute; for example, shipping followed by receiving, or a more complex sequence such as loading → departing → transporting → arriving → unloading → accepting. The `ReadPoint` and `BusinessLocation` in the "where" dimension of these EPCIS events indicate the known physical location at each step of the process. `Source` and `Destination`, in contrast, may be used to indicate the parties and/or location that are the intended endpoints of the business transfer. In a multi-step business transfer, some or all of the EPCIS events may carry `Source` and `Destination`, and the information would be the same for all events in a given transfer.

`Source` and `Destination` provide a standardised way to indicate the parties and/or physical locations involved in the transfer, complementing the business transaction information (e.g., purchase orders, invoices, etc.) that may be referred to by `BusinessTransaction` elements.

A source or destination is described in EPCIS by a structured type consisting of a pair of identifiers, as follows.

Field	Type	Description
type	SourceDestTypeID	(Optional in BizTransaction, required in SourceOrDestination, and optional in SensorReport). Indicates the kind of BizTransaction document (e.g., po), role of SourceOrDestination (e.g., owning_party), or kind of measurement in SensorReport (e.g., Mass).
source or destination	SourceDestID	An identifier that denotes a specific source or destination.

The two vocabulary types `SourceDestTypeID`, and `SourceDestID` are defined in the sections below.

7.3.6.4.1 Source/Destination type

`SourceDestTypeID` is a vocabulary whose elements denote a specific type of business transfer source or destination. An example is an identifier that denotes "owning party." `SourceDestTypeID` is an example of a Standard Vocabulary as defined in section 6.2.

Non-Normative: Explanation: Using an extensible vocabulary for source/destination type identifiers allows GS1 standards to define some common terms such as "owning party" while allowing for industry groups and individual end-users to define their own terms. Master data may provide additional information.

This specification defines no master data attributes for source/destination type identifiers.

7.3.6.4.2 Source/Destination ID

`SourceDestID` is a vocabulary whose elements denote specific sources and destinations. An example is an identifier that denotes "Acme Corporation (an owning party)." `SourceDestID` is a User Vocabulary as defined in section 6.2.

Note:

The `PartyID` type is a supertype of `SourceDestID`. The `urn:epcglobal:epcis:vtype:Location` URI may be used to specify a single vocabulary containing identifiers that may populate the source or destination field(s) of associated EPCIS events.

Non-Normative: Explanation: URIs are used to provide extensibility and a convenient way for organisations to distinguish one kind of source or destination identifier from another.

7.3.7 The "How" dimension

This section defines the data fields and the expected value types and data types used in the "How" dimension.

7.3.7.1 SensorElement

A `SensorElement` is a structure that contains an optional `sensorMetadata` element and one or several `sensorReport` elements, described as follows:

Field	Type	Description
`sensorMetadata`	`SensorMetadata`	(Optional) An element containing one or several metadata attributes, which are applicable to all `sensorReport` elements that are part of the same `sensorElement`.
`sensorReport`	`SensorReport`	An element containing one or several attributes that pertain to a specific sensor observation.

For the sake of compactness, all elements contained in either the `sensorMetadata` or `sensorReport` element are inline attributes.

The following rules apply:

1. A `sensorElement` parent element MAY contain exactly one `sensorMetadata` element and SHALL contain one or more `sensorReport` elements.
2. The values of any inline attributes specified within the `sensorMetadata` element are assumed to apply for all `sensorReport` elements contained within that same `sensorElement`.
3. The following inline attributes are permitted to appear both in a `sensorMetadata` as well as in a `sensorReport` element: `dataProcessingMethod`, `deviceID`, `deviceMetadata`, `rawData`, and `time` (which are specified in section 7.3.7.1.1). If they are present in a `sensorElement` container, they SHALL either be specified in the `sensorMetadata` element or in the `sensorReport` element(s) and SHALL NOT appear in both.

Non-normative: Explanation: Even though it would technically be feasible, EPCIS SHOULD NOT be used to accommodate **raw** sensor data unless there is a strong reason to do so. The added value of the `sensorElement` in EPCIS consists in the abstraction from raw sensor data and provisioning of aggregated, business-oriented data to accessing applications. For instance, instead of capturing thousands of time-stamped datasets, it is often far more appropriate and efficient to only indicate the range of values of a given sensor property within a given period of time. For that purpose, an EPCIS capturing application would only need to populate four fields: `minValue`, `maxValue`, `startTime`, and `endTime`. Even if there is a business need to have the ability to access the underlying raw sensor data, it is neither required nor advisable to include raw data in the EPCIS event. Instead, it is advisable to include a Web URI in the `rawData` element, pointing to a resource through which clients can access the underlying raw sensor data.

7.3.7.1.1 SensorMetadata

A `SensorMetadata` element contains a number of inline attributes, defined as follows.

Field	Type	Description
time	xsd:dateTimeStamp	(Optional) The actual point in time of an observation as transmitted by a sensor device. If present, the time value SHALL be less (earlier) than or equal to the eventTime value. It SHALL also contain a time zone offset value. The coordination and integrity of distributed computing requires time synchronisation of EPCIS events conveying sensor data. Therefore, when populating the time, startTime, and endTime field, EPCIS capture applications SHOULD apply established time synchronisation protocols such as IEEE 1588-2008 , which provides a standard method to synchronise device clocks in a network. Note: The eventTime applies to the completion of a business step, not a sensor observation. For instance, for a receiving event accommodating a sensor element, the eventTime indicates when goods were received – it gives no information when certain conditions (e.g. a specific temperature value) held true. In some circumstances, event and sensor observation times may correspond though (e.g. if a quality inspector checks certain properties of goods). In such cases, indicating the eventTime may be sufficient.
startTime	xsd:dateTimeStamp	(Optional) The lowest (earliest) value of a given observation period as transmitted by a sensor device. If present, the startTime SHALL be less (earlier) than the eventTime value and the endTime value.
endTime	xsd:dateTimeStamp	(Optional) The highest (most recent) value of a given observation period, as transmitted by a sensor device. If present, the endTime SHALL be less (earlier) than or equal to the eventTime value.
deviceID	EPC	(Optional) Device from which the sensor data originates.
deviceMetadata	ResourceID	(Optional) Storage location of an electronic document accommodating metadata of the device from which the sensor data originates.
rawData	ResourceID	(Optional) Storage/service location of the raw sensor data on which the aggregated/business-oriented data contained in the sensorElement is based.
dataProcessingMethod	ResourceID	(Optional) Storage location of an electronic document accommodating the data processing method of the contained sensor data, if applicable. For instance, before sensor data is captured in an EPCIS event, the latter might be redacted or refined by means of specific algorithms.

Field	Type	Description
bizRules	ResourceID	(Optional) Storage location of an electronic document accommodating product- or application-specific business rules on which basis the EPCIS event was triggered. For instance, an EPCIS capturing application might only trigger an event if the 'what' dimension contains a certain product class (e.g. GTIN of a cold-storage pharmaceutical product) AND a certain temperature threshold was exceeded. Note: In contrast to a business transaction, a business rules file is not a standard-defined, interoperably communicated business document such as an invoice. In addition, the set of rules typically will only change in an infrequent manner.

7.3.7.1.2 SensorReport

A `sensorReport` element contains a number of inline attributes, defined as follows.

Field	Type	Description
type	SensorPropertyTypeID	An identifier that indicates what kind of property the `SensorReport` element pertains to. Expected values for `type` can be MeasurementType or a custom URI. Sensor measurement types SHALL be expressed using either URIs or Compact URI Expressions (CURIEs), as follows: • `https://gs1.org/voc/X` • `gs1: X` where the *X* part is a string as specified in CBV section 7.6.3
exception	gs1:SensorAlertType	(Required if there is no `type` field in `SensorReport`) An identifier that indicates what kind of exception this `SensorReport` denotes.
deviceID	EPC	See previous section.
deviceMetadata	ResourceID	See previous section.
rawData	ResourceID	See previous section.
dataProcessingMethod	ResourceID	See previous section.
time	xsd:dateTimeStamp	See previous section.
microorganism	MicroorganismID	(Optional) Identifies a specific microorganism species. If `microorganism` is present, a `SensorReport` element SHALL NOT include `chemicalSubstance`.
chemicalSubstance	ChemicalSubstanceID	(Optional) Identifies a specific chemical substance. If `chemicalSubstance` is present, a `SensorReport` element SHALL NOT include `microorganism`.

Field	Type	Description
value	xsd:double	(Optional) Value of the property specified by the type as part of the sensorReport element. If a chemicalSubstance or microorganism field is present, the value indicates the detected measurement (e.g., of molar mass or concentration) of a chemical substance or microorganism in the physical objects specified in the 'What' dimension. If a time field is present, the value SHALL pertain to this point in time. Otherwise, value SHALL refer to the eventTime. *NOTE:* xsd:float *supports values with positive or negative exponents, whereas* xsd:decimal *supports only values with negative exponents.* For example, a sensor pressure value in pascals may have positive exponents.
component	cbv:Comp	(Optional) Some measurement types (e.g. force, pressure) are vectors (having magnitude and direction in space), whereas others (e.g. temperature, absolute humidity) are scalars. The components of a vector in a coordinate system can be specified by setting the value of component to indicate which vector component has magnitude indicated by the value parameter. The SensorReport element is then repeated as two or three instances to express each component of the vector in two or three dimensions, the pair or trio of SensorReport elements sharing the same values for all fields except value, component and uom (which may differ for each vector component). If present, the SensorReport element MUST also include the value and uom fields. Sensor measurement types SHALL be expressed using either URIs or Compact URI Expressions (CURIEs), as follows: • https://gs1.org/voc/X • gs1:X where the *X* part is a string as specified in **CBV section 7.8** Enumeration list: • X, Y, Z (cartesian coordinates in 2 or 3 dimensions) • axialDistance, azimuth, height (cylindrical polars) • sphericalRadius, azimuth, inclination / polarAngle, elevationAngle (spherical polars) • latitude, longitude, elevation / altitude • easting, northing
stringValue	String	(Optional) The String value of the property specified by the type and/or exception field as part of the sensorReport element. If a time field is present, the String value SHALL pertain to this point in time. Otherwise, it SHALL refer to the eventTime.

Field	Type	Description
booleanValue	Boolean	(Optional) The Boolean value of the property specified by the type and/or exception field as part of the sensorReport element. If a time field is present, the Boolean value SHALL pertain to this point in time. Otherwise, it SHALL refer to the eventTime. If a chemicalSubstance or microorganism field is included, booleanValue=true means that this chemical substance/microorganism is present, while booleanValue=false means that this chemical substance/microorganism is absent, whereas value can be used to specify the concentration; a non-zero concentration value is incompatible with booleanValue=false.
hexBinaryValue	HexBinary	(Optional) The HexBinary value of the property specified by the type and/or exception field as part of the sensorReport element. If a time field is present, the HexBinary value SHALL pertain to this point in time. Otherwise, it SHALL refer to the eventTime.
uriValue	AnyURI	(Optional) The URI value of the property specified by the type and/or exception field as part of the sensorReport element. If a time field is present, the URI value SHALL pertain to this point in time. Otherwise, it SHALL refer to the eventTime.
minValue	xsd:double	(Optional) Minimum quantitative value of the property specified by type , as part of the sensorReport element. If a startTime and endTime field is present, the minValue SHALL pertain to the resulting period. Otherwise, minValue SHALL pertain to the business process step the EPCIS event captures.
maxValue	xsd:double	(Optional) Maximum quantitative value of the property specified by type, as part of the SensorReport element. If startTime and endTime fields are present, the maxValue SHALL pertain to the resulting period. Otherwise, maxValue SHALL pertain to the business process step the EPCIS event captures.
meanValue	xsd:double	(Optional) The arithmetic mean of the values of the property specified by the type as part of the sensorReport element. If a startTime and endTime field is present, the meanValue SHALL pertain to the resulting period. Otherwise, it SHALL pertain to the business process step the EPCIS event captures.
sDev	xsd:double	(Optional) Standard deviation of the values of the property specified by type, as part of the sensorReport element. sDev SHALL only be used in conjunction with the field meanValue.
percRank	xsd:double	(Optional) Percentile rank, signifying the percentage of observations in a frequency distribution that are equal to or lower than it. percRank SHALL only be used in conjunction with the field percValue.
percValue	xsd:double	(Optional) The percentile value, at or below which a given percentage of observations (as specified by percRank) may be found. percValue SHALL only be used in conjunction with the field percRank.

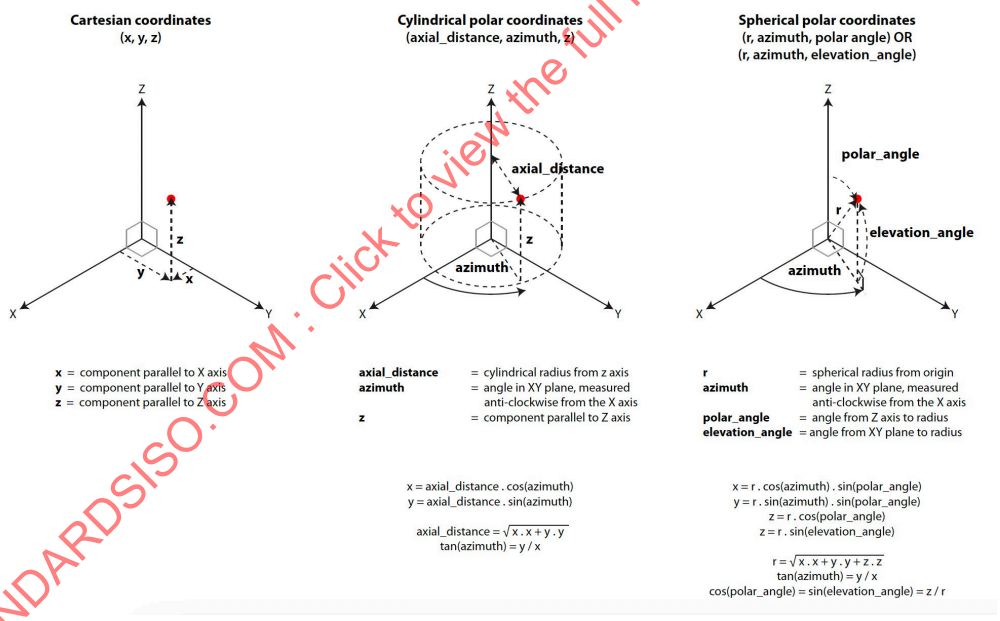
Field	Type	Description
`uom`	`UOM`	(Optional) 2- or 3-character code for a physical unit specified in the `Common Code` column of UN/CEFACT Recommendation 20. Defines a unit of measure by which the specified value(s) should be interpreted. If there is no `value`, `minValue`, maxValue, `meanValue` or `percValue` field present, `uom` SHALL be omitted. If present, `uom` should correlate with the specified `type` (`epcis:measurementType`), e.g., `uom` "KGM" can only be used with `type` "`Mass`".
`coordinateReferenceSystem`	`anyURI`	(Optional) A URI identifying the Coordinate Reference System. If omitted, the World Geodetic System (WGS 84) [WGS84] is assumed to apply.

NOTE on extensibility by users/vendors:

Because of `<xsd:anyAttribute processContents="lax"/>` within XSD validation of SensorReport, any other (user-defined) attributes can be included / tolerated and will validate, even if they use other data types.

7.3.7.1.3 Coordinate reference systems (CRS)

Figure 7-5 Coordinate reference systems



Non-Normative: Note: In addition to the EPCIS standard attributes as defined above, organisations may wish to provide further sensor-related context information. In such a case, they may populate the sensorMetadata and sensorReport element with additional attributes. The latter SHOULD be standardised to the greatest extent possible, through the usage of linked data attributes as specified in [SSN].

7.3.7.1.4 Sensor property type

`SensorPropertyTypeID` is a *standard vocabulary* type (see section 7.2.3) whose elements denote specific sensor properties. An example is an identifier that denotes 'temperature'.

7.3.7.1.5 UOM

The `uom` field qualifies the unit of measure of one or several double-precision floating point values that are part of the same `sensorReport` element and pertain to the property indicated by the `type` attribute. When a `uom` field is present, its value SHALL be the 2- or 3-character code for a physical unit specified in the "Common Code" column of UN/CEFACT Recommendation 20 [CEFACT20]. Moreover, the code SHALL NOT be marked as "X" (deleted) or "D" (deprecated).

7.3.7.1.6 Microorganism ID

`MicroorganismID` is a *user vocabulary* type (see section 7.2.3) whose elements denote specific microorganisms. An example is an identifier that denotes 'Listeria monocytogenes'.

7.3.7.1.7 Chemical Substance ID

`ChemicalSubstanceID` is a *user vocabulary* type (see section 7.2.3) whose elements denote specific chemical substances. An example is an identifier that denotes "sucrose".

7.3.7.1.8 Resource ID

`ResourceID` is a *user vocabulary* type (see section 7.2.3) whose elements denote specific electronic information resources. An example is an identifier that denotes a product information page provided by the manufacturer of a sensor device.

7.3.8 Instance/Lot master data (ILMD)

Instance/Lot master data (ILMD) is data that describes a specific instance of a physical or digital object, or a specific batch/lot of objects that are produced in batches/lots. ILMD consists of a set of descriptive attributes that provide information about one or more specific objects or lots. It is similar to ordinary master data, which also consists of a set of descriptive attributes that provide information about objects. But whereas master data attributes have the same values for a large class of objects, (e.g., for all objects having a given GTIN), the values of ILMD attributes may be different for much smaller groupings of objects (e.g., a single batch or lot), and may be different for each object (i.e., different for each instance). Note that the term 'attribute' typically corresponds to a fieldname or Linked Data property.

An example of a master data attribute is the weight and physical dimensions of trade items identified by a specific GTIN. These values are the same for all items sharing that GTIN. An example of ILMD is the expiration date of a perishable trade item. Unlike master data, the expiration date is not the same for all trade items having the same GTIN; in principle, each may have a different expiration date depending on when it is manufactured. Other examples of ILMD include date of manufacture, place of manufacture, weight and other physical dimensions of a variable-measure trade item, harvest information for farm products, and so on.

ILMD, like ordinary master data, is intended to be static over the life of the object. For example, the expiration date of a perishable trade item or the weight of a variable-measure item does not change over the life of the trade item, even though different trade items having the same GTIN may have different values for expiration date and weight. ILMD is *not* to be used to represent information that changes over the life of an object, for example, the current temperature of an object as it moves through the supply chain.

While there exist standards (such as GDSN) for the registration and dissemination of ordinary master data through the supply chain, standards and systems for dissemination of ILMD do not yet exist. For this reason, EPCIS allows ILMD to be carried directly in certain EPCIS events. This feature should only be used when no separate system exists for dissemination of ILMD.

ILMD for a specific object is defined when the object comes into existence. Therefore, ILMD may only be included in `ObjectEvent`s with action `ADD` (section 7.4.1.2), and in `TransformationEvent`s (section 7.4.5). In the case of a `TransformationEvent`, ILMD applies to the outputs of the transformation, not to the inputs.

The structure of ILMD defined in this EPCIS standard consists of a set of named attributes, with values of any type. In the XML binding (section 9.5), the XML schema provides for an unbounded list of XML elements having any element name and content. Other documents layered on top of EPCIS may define specific ILMD data elements; see section 6.3. In this way, ILMD is similar to event-level EPCIS extensions, but is separate in order to emphasise that ILMD applies for the entire life of objects, whereas an event-level EPCIS extension only applies to that specific event.

7.4 Core event types module – events

7.4.1 EPCISEvent

`EPCISEvent` is a common base type for all EPCIS events. All of the more specific event types in the following sections are subclasses of `EPCISEvent`.

This common base type only has the following fields.

Field	Type	Description
`eventTime`	`xsd:dateTimeStamp`	The date and time at which the EPCIS Capturing Applications asserts the event occurred.
`recordTime`	`xsd:dateTimeStamp`	(Optional) The date and time at which this event was recorded by an EPCIS Repository. This field SHALL be ignored when an event is presented to the EPCIS Capture Interface, and SHALL be present when an event is retrieved through the EPCIS Query Interfaces. The `recordTime` does not describe anything about the real-world event, but is rather a bookkeeping mechanism that plays a role in the interpretation of standing queries as specified in section 8.2.5.2.
`eventTimeZoneOffset`	`xsd:string`	The time zone offset in effect at the time and place the event occurred, expressed as an offset from UTC. The value of this field SHALL be a string consisting of the character '+' or the character '−', followed by two digits whose value is within the range `00` through `14` (inclusive), followed by a colon character '`:`', followed by two digits whose value is within the range `00` through `59` (inclusive), except that if the value of the first two digits is `14`, the value of the second two digits must be `00`. For example, the value `+05:30` specifies that where the event occurred, local time was five hours and 30 minutes later than UTC (that is, midnight UTC was 5:30am local time).

Field	Type	Description
`eventID`	`EventID`	(Optional) An identifier for this event as specified by the capturing application, globally unique across all events other than error declarations. "Globally unique" means different from the `eventID` on any other EPCIS event across any implementation of EPCIS, not merely across the events captured by a single capturing application or by a single capture server. (The Core Business Vocabulary standard [CBV2.0] specifies the option of a UUID URI or EPCIS Event Hash ID for this purpose.) Note that in the case of an error declaration, the event ID will be equal to the event ID of the erroneous event (or null if the event ID of the erroneous event is null), and in that sense is not unique. See section 7.4.1.2.
`errorDeclaration`	`ErrorDeclaration`	(Optional) If present, indicates that this event serves to assert that the assertions made by a prior event are in error. See section 7.4.1.2.
`certificationInfo`	`gs1:CertificationDetails`	(Optional) If present, specifies a URL at which certification details can be found. Machine-interpretable certification details may be expressed using properties within the gs1:CertificationDetails class of the GS1 Web Vocabulary

7.4.1.1 Explanation of eventTimeZoneOffset (Non-Normative)

The `eventTimeZoneOffset` field is *not* necessary to understand at what moment in time an event occurred. This is because the `eventTime` field is of type `Time`, defined in section 7.3 to be a "date and time in a time zone-independent manner." For example, in the XML binding (section 9.5) the `eventTime` field is represented as an element of type `xsd:dateTimeStamp`, and section 9.5 further stipulates that the XML must include a time zone specifier. Therefore, the XML for `eventTime` unambiguously identifies a moment in absolute time, and it is not necessary to consult `eventTimeZoneOffset` to understand what moment in time that is.

The purpose of `eventTimeZoneOffset` is to provide additional business context about the event, namely to identify what time zone offset was in effect at the time and place the event was captured. This information may be useful, for example, to determine whether an event took place during business hours, to present the event to a human in a format consistent with local time, and so on. The local time zone offset information is *not* necessarily available from `eventTime`, because there is no requirement that the time zone specifier in the XML representation of `eventTime` be the local time zone offset where the event was captured. For example, an event taking place at 8:00am US Eastern Standard Time could have an XML `eventTime` field that is written `08:00-05:00` (using US Eastern Standard Time), or `13:00Z` (using UTC), or even `07:00-06:00` (using US Central Standard Time). Moreover, XML processors are not required by [XSD2] to retain and present to applications the time zone specifier that was part of the `xsd:dateTimeStamp` field, and so the time zone specifier in the `eventTime` field might not be available to applications at all. Similar considerations would apply for other (non-XML) bindings of the Core Event Types module. For example, a hypothetical binary binding might represent `Time` values as a millisecond offset relative to midnight UTC on January 1, 1970 – again, unambiguously identifying

a moment in absolute time, but not providing any information about the local time zone. For these reasons, `eventTimeZoneOffset` is provided as an additional event field.

7.4.1.2 ErrorDeclaration

When an event contains an `ErrorDeclaration` element, it indicates that this event has special semantics: instead of the normal semantics which assert that various things happened and that various things are true following the event, the semantics of this event assert that those prior assertions are in error. An event containing an `ErrorDeclaration` element SHALL be otherwise identical to a prior event, "otherwise identical" meaning that all fields of the event other than the `ErrorDeclaration` element and the value of `recordTime` are exactly equal to the prior event. (Note that includes the `eventID` field: the `eventID` of the error declaration will be equal to the `eventID` of the prior event or null if the `eventID` of the prior event is null. This is the sole case where the same non-null `eventID` may appear in two events.) The semantics of an event containing the `ErrorDeclaration` element are that all assertions implied by the prior event are considered to be erroneous, as of the specified `declarationTime`. The prior event is not modified in any way, and subsequent queries will return both the prior event and the error declaration.

An `ErrorDeclaration` element contains the following fields:

Field	Type	Description
`declarationTime`	`xsd:dateTimeStamp`	The date and time at which the declaration of error is made. (Note that the `eventTime` of this event must match the `eventTime` of the prior event being declared erroneous, so the `declarationTime` field is required to indicate the time at which this event is asserted.)
`reason`	`ErrorReasonID`	(Optional) An element from a standard vocabulary that specifies the reason the prior event is considered erroneous.
`correctiveEventIDs`	`List<EventID>`	(Optional) If present, indicates that the events having the specified URIs as the value of their `eventID` fields are to be considered as "corrections" to the event declared erroneous by this event. This provides a means to link an error declaration event to one or more events that are intended to replace the erroneous event.

An `ErrorDeclaration` element SHOULD NOT be used if there is a way to model the real-world situation as an ordinary event (that is, using an event that does not contain an `ErrorDeclaration` element).

7.4.1.2.1 Use of error declarations (Non-Normative)

An EPCIS event records the completion of a step of a business process. A business process is modelled by breaking it down into a series of steps, and modelling each as an EPCIS event. The net effect is that the collection of all events pertaining to a specific object (often referred to as a "trace") should correctly indicate the history and current state of that object, by interpreting the events according to the semantics specified in this standard and relevant vocabulary standards.

Sometimes, it is discovered that an event recorded earlier does not accurately reflect what happened in the real world. In such cases, as noted in section 6.1, earlier events are never deleted or modified. Instead, additional events are recorded whose effect is that the complete trace (including the new events and all prior events including the incorrect event) accurately reflects the history and current state, as stated in the above principle.

The preferred way to arrive at the additional events is to recognise that the discovery of an erroneous event and its remediation is itself a business process which can be

modelled by creating suitable EPCIS events. In most situations, this is done using EPCIS events from the Core Event Types Module as specified in section 7.4, using suitable vocabulary.

Example 1: Company X records an EPCIS event asserting that serial numbers 101, 102, and 103 of some product were shipped to Company Y. Company Y receives the shipment and finds serial number 104 in addition to serial numbers 101, 102, 103. In discussion with Company X, it is agreed that serial 104 was indeed shipped and that the shipping event was in error. Remediation: Company X records a new EPCIS event asserting that serial number 104 was shipped, with similar contextual information as the original event.

Example 2: Company X records an EPCIS event asserting that serial numbers 101, 102, and 103 of some product were shipped to Company Y. Company Y receives the shipment and finds only serial numbers 101, 102. In discussion with Company X, it is agreed that serial 103 was not shipped but remains in Company X's inventory. They agree to reverse the billing for the third product. Remediation: Company X records a new EPCIS event asserting that the shipment of serial 103 is voided.

In the first example, the additional event uses the same business vocabulary as the first. In the second example, vocabulary specifically associated with the process of voiding a shipment is used, but it is still "ordinary" EPCIS semantics in the sense that it models the completion of a well-defined business process step. This reflects the reality that the act remediation is itself a business process, and so may be modelled as an EPCIS event.

In some situations, it either is not possible (or is highly undesirable) to remediate the history of an object by creating a new EPCIS event with ordinary semantics (that is, with the semantics specified in section 7.4).

Example 3: Company X records an EPCIS event to assert that serial number 101 of product X was destroyed. This event is an Object Event as specified in section 7.4.2 with action = DELETE. Later it is discovered that serial 101 is still in storage, not destroyed. An ordinary event cannot be used to amend the history, because the semantics of action DELETE for an Object Event specify that "the objects … SHOULD NOT appear in subsequent events."

Example 4: Company X records an EPCIS event asserting that several products have been shipped, indicating Purchase Order 123 as a business transaction in the "why" dimension. Company Y receives the products and records a receiving event. Only then it is discovered that the purchase order reference in the shipping event is wrong: it says PO 456 instead of 123. This could be remediated using ordinary EPCIS events by Company X recording a "cancel shipment" event followed by a "shipping" event with the correct PO #. But this is rather undesirable from the perspective of the overall trace, especially given that there is already a receiving event.

To accommodate such situations, the Core Event Types Module provides a mechanism to assert that the assertions made by a prior event are in error. These semantics may only be used when an event specifies exactly the same conditions as a prior ordinary event so that the assertion of error can be correlated to the prior event. Such an event is termed an "error declaration event."

In Example 3 above, the error declaration event would imply that serial number 101 of product X was not destroyed. In Example 4 above, the error declaration event would imply that a shipment with PO 123 as the context did not occur, and an additional event (the "corrective event") would say that a shipment with PO 456 did occur. This is rather similar to modelling Example 4 using a "cancel shipment" event, except that instead of asserting a shipment was carried out under PO 123 then cancelled, the error declaration event simply asserts that the PO 123 assertion was erroneous.

An error declaration event is constructed by including an `ErrorDeclaration` section. Specifically, given Event E1, an error declaration event E2 whose effect is to declare the assertions of E1 to be in error is an event structure whose content is identical to E1, but with the `ErrorDeclaration` element included. For example, the error declaration for the "destroying" event in Example 3 is also an Object Event with action = DELETE, but with the `ErrorDeclaration` element included. In general, to declare event E to be in error, a new event is recorded that is identical to event E except that the `ErrorDeclaration` element is also included (and the record time will be different).

There are three reasons why error declaration events in EPCIS are expressed this way. One, an event ID is not required to indicate the erroneous event, which in turn implies it is not necessary to include an event ID on every event to provide for possible error declaration in the future. Event IDs are available to link an error declaration event to a corrective event, but it is never necessary to use event IDs. Two, any EPCIS query that matches an event will also match an error declaration for that event, if it exists. This means that EPCIS Accessing Applications require no special logic to become aware of error declarations, if they exist. Three, if an EPCIS Accessing Application receives an error declaration event and for some reason does *not* have a copy of the original (erroneous) event, it is not necessary to retrieve the original event as every bit of information in that event is also present in the error declaration event.

7.4.1.2.2 Matching an error declaration to the original event (non-normative)

As discussed in section 7.4.1.2, an error declaration event has identical content to the original (erroneous) event, with the exception of the `ErrorDeclaration` element itself and the record time. One of the benefits of this approach is that when an EPCIS Accessing Application encounters an error declaration, it is not necessary to retrieve the original (erroneous) event, as all of the information in that event is also present in the error declaration event which the EPCIS Accessing Application already has.

Nevertheless, there may be situations in which an EPCIS Accessing Application or EPCIS Capturing Application wishes to confirm the existence of the original (erroneous) event by querying for it. The only way to recognise that an event is the original event matching an error declaration is to confirm that all data elements in the events (save the `ErrorDeclaration` element and record time) match. See [EPCISGuideline] for suggested approaches for querying in this situation.

7.4.2 ObjectEvent (subclass of EPCISEvent)

An `ObjectEvent` captures information about an event pertaining to one or more physical or digital objects identified by instance-level (EPC) or class-level (EPC Class) identifiers. Most `ObjectEvent`s are envisioned to represent actual observations of objects, but strictly speaking it can be used for any event a Capturing Application wants to assert about objects, including for example capturing the fact that an expected observation failed to occur.

While more than one EPC and/or EPC Class may appear in an `ObjectEvent`, no relationship or association between those objects is implied other than the coincidence of having experienced identical events in the real world.

The `action` field of an `ObjectEvent` describes the event's relationship to the lifecycle of the objects and their identifiers named in the event. Specifically:

`action` value	Meaning
`ADD`	The objects identified in the event have been commissioned as part of this event. For objects identified by EPCs (instance-level identifiers), the EPC(s) have been issued and associated with an object (s) for the first time. For objects identified by EPC Classes (class-level identifiers), the specified quantities of EPC Classes identified in the event have been created (though other instances of those same classes may have existed prior this event, and additional instances may be created subsequent to this event).
`OBSERVE`	The event represents a simple observation of the objects identified in the event, not their commissioning or decommissioning.
`DELETE`	The objects identified in the event have been decommissioned as part of this event. For objects identified by EPCs (instance-level identifiers), the EPC(s) do not exist subsequent to the event and SHOULD NOT be observed again. For objects identified by EPC Classes (class-level identifiers), the specified quantities of EPC Classes identified in the event have ceased to exist (though other instances of those same classes may continue to exist subsequent to this event, and additional instances may be have ceased to exist prior this event).

An `ObjectEvent` has the following fields:

Field	Type	Description

`eventTime`
`recordTime`
`eventTimeZoneOffset`
	(Inherited from `EPCISEvent`; see section 7.4.1)	

`epcList`	`List<EPC>`	(Optional) An unordered list of one or more EPCs naming specific objects to which the event pertained. See section 7.3.3.2 An ObjectEvent SHALL contain either a non-empty `epcList`, a non-empty `quantityList`, or both. The only permissible exception is if the object of observation is a physical location – in this case, the ObjectEvent SHALL contain a `sensorElement` and a non-empty `readPoint` populated with a physical location ID.
`quantityList`	`List<QuantityElement>`	(Optional) An unordered list of one or more QuantityElements identifying (at the class level) objects to which the event pertained. An ObjectEvent SHALL contain either a non-empty `epcList`, a non-empty `quantityList`, or both. The only permissible exception is if the object of observation is a physical location – in this case, the ObjectEvent SHALL contain a `sensorElement` and a non-empty `readPoint` populated with a physical location ID.
`Action`	`Action`	How this event relates to the lifecycle of the EPCs named in this event. See above for more detail.
`bizStep`	`BusinessStepID`	(Optional) The business step of which this event was a part.
`disposition`	`DispositionID`	(Optional) The business condition of the objects, in the `epcList` and `quantityList`, presumed to hold true until contradicted by a subsequent event.
`persistentDisposition`	`PersistentDisposition`	(Optional) One or more business conditions of the objects, in the `epcList` and `quantityList`. Each persistentDisposition is explicitly `set` and `unset` independently of other `persistentDisposition` values. The `set` field within `persistentDisposition` may specify a list of `persistentDisposition` URI values to be set. The `unset` field within `persistentDisposition` may specify a list of `persistentDisposition` URI values to be unset (revoked).

Field	Type	Description
`set`	`DispositionID`	(Optional, multivalued) If used in `PersistentDisposition`, specifies `Disposition` (`cbv:Disp`) values to be **`set` "persistently"**, i.e., **until they are explicitly `unset`**.
`unset`	`DispositionID`	(Optional, multivalued) If used in `PersistentDisposition`, specifies `Disposition` (`cbv:Disp`) values to be **`unset` "persistently"**, i.e., **until they are explicitly `set`**.
`readPoint`	`ReadPointID`	(Optional) The read point at which the event took place.
`bizLocation`	`BusinessLocationID`	(Optional) The business location where the objects associated with the EPCs may be found, until contradicted by a subsequent event.
`bizTransactionList`	Unordered list of zero or more `BusinessTransaction` instances	(Optional) An unordered list of business transactions that define the context of this event.
`sourceList`	`List<Source>`	(Optional) An unordered list of `Source` elements (section 7.3.6.4) that provide context about the originating endpoint of a business transfer of which this event is a part.
`destinationList`	`List<Destination>`	(Optional) An unordered list of `Destination` elements (section 7.3.6.4) that provide context about the terminating endpoint of a business transfer of which this event is a part.
`ilmd`	`ILMD`	(Optional) Instance/Lot master data (section 7.3.8) that describes the objects created during this event. An `ObjectEvent` SHALL NOT contain `ilmd` if `action` is `OBSERVE` or `DELETE`.
`sensorElementList`	`List<sensorElement>`	(Optional) An unordered list of one or more `sensorElement`s (section 7.3.7).

Note that in the XML binding (section 9.3), `quantityList`, `sourceList`, `destinationList`, `ilmd` and `sensorElementList` appear in the standard extension area, to maintain backward-compatibility with EPCIS 1.0, 1.1 and 1.2.

Retrospective semantics:

- An event described by `bizStep` (and any other fields) took place with respect to the objects identified by `epcList` and `quantityList` at `eventTime` at location `readPoint`.
- (If `action` is `ADD`) The EPCs in `epcList` were commissioned (issued for the first time).

- (If `action` is `ADD`) The specified quantities of EPC Class instances in `quantityList` were created (or an unknown quantity, for each `QuantityElement` in which the `quantity` value is omitted).
- (If `action` is `DELETE`) The EPCs in `epcList` were decommissioned (retired from future use).
- (If `action` is `DELETE`) The specified quantities of EPC Class instances in `quantityList` ceased to exist (or an unknown quantity, for each `QuantityElement` in which the `quantity` value is omitted).
- (If `action` is `ADD` and a non-empty `bizTransactionList` is specified) An association exists between the business transactions enumerated in `bizTransactionList` and the objects identified in `epcList` and `quantityList`.
- (If `action` is `OBSERVE` and a non-empty `bizTransactionList` is specified) This event took place within the context of the business transactions enumerated in `bizTransactionList`.
- (If `action` is `DELETE` and a non-empty `bizTransactionList` is specified) This event took place within the context of the business transactions enumerated in `bizTransactionList`.
- (If `sourceList` is non-empty) This event took place within the context of a business transfer whose originating endpoint is described by the sources enumerated in `sourceList`.
- (If `destinationList` is non-empty) This event took place within the context of a business transfer whose terminating endpoint is described by the destinations enumerated in `destinationList`.
- (If `sensorElementList` is non-empty) This event took place in the context of the sensor observation specified in the `sensorElementList` at `time` or during `startTime` and `endTime` (or at `eventTime` when `time`, `startTime` and `endTime` are omitted). All values pertain to the objects identified by `epcList` and `quantityList` (or the physical location indicated in the `readPoint` when both `epcList` and `quantityList` are omitted).

Prospective semantics:

- (If `action` is `ADD`) The objects identified by the instance-level identifiers in `epcList` may appear in subsequent events.
- (If `action` is `ADD`) The objects identified by the class-level identifiers in `quantityList` may appear in subsequent events.
- (If `action` is `DELETE`) The objects identified by the instance-level identifiers in `epcList` SHOULD NOT appear in subsequent events.
- (If `action` is `DELETE`) The total population of objects identified by the class-level identifiers in `quantityList` that may appear in subsequent events has been reduced by the quantities specified in `quantityList` (or by an unknown quantity, for each `QuantityElement` in which the `quantity` value is omitted).
- (If `disposition` is specified) The business condition of the objects identified by `epcList` and `quantityList` is as described by `disposition`.
- (If `disposition` is omitted) The `business` condition of the objects identified by `epcList` and `quantityList` is unchanged.
- (If a specific `persistentDisposition` is specified as `set`) The persistent business condition(s) of the objects identified by `epcList` and `quantityList` is as set by `persistentDisposition`.
- (If `persistentDisposition` is omitted) The persistent business condition(s) of the objects identified by `epcList` and `quantityList` as previously `set` or `unset` by `persistentDisposition is unchanged`.

- (If a specific `persistentDisposition` value is specified as `unset`) The specific persistent `business` condition of the objects identified by `epcList` and `quantityList` is unset (i.e., revoked).
- (If `bizLocation` is specified) The objects identified by `epcList` and `quantityList` are at business location `bizLocation`.
- (If `bizLocation` is omitted) The business location of the objects identified by `epcList` and `quantityList` might be unknown, unless the business location can be inferred from previous events.
- (If `action` is `ADD` and `ilmd` is non-empty) The objects identified by `epcList` and `quantityList` are described by the attributes in `ilmd`.
- (If `action` is `ADD` and a non-empty `bizTransactionList` is specified) An association exists between the business transactions enumerated in `bizTransactionList` and the objects identified in `epcList` and `quantityList`.

Non-Normative: Explanation: In the case where action is ADD and a non-empty `bizTransactionList` is specified, the semantic effect is equivalent to having an ObjectEvent with no `bizTransactionList` together with a TransactionEvent having the `bizTransactionList` and all the same field values as the ObjectEvent. Note, however, that an ObjectEvent with a non-empty `bizTransactionList` does not cause a TransactionEvent to be returned from a query.

7.4.3 AggregationEvent (subclass of EPCISEvent)

The event type `AggregationEvent` describes events that apply to objects that have been aggregated to one another. In such an event, there is a set of "contained" objects that have been aggregated within a "containing" entity that's meant to identify the aggregation itself.

This event type is intended to be used for "aggregations," in which there is a strong physical relationship between the containing and the contained objects such that they will all occupy the same location at the same time, until such time as they are disaggregated. An example of an aggregation is where cases are loaded onto a pallet and carried as a unit. The `AggregationEvent` type is not intended for weaker relationships such as two pallets that are part of the same shipment, but where the pallets might not always be in exactly the same place at the same time. (The `TransactionEvent` may be appropriate for such circumstances.) More specific semantics may be specified depending on the Business Step.

The `Action` field of an `AggregationEvent` describes the event's relationship to the lifecycle of the aggregation. Specifically:

`Action` value	Meaning
`ADD`	The objects identified in the child list have been aggregated to the parent during this event. This includes situations where the aggregation is created for the first time, as well as when new children are added to an existing aggregate.
`OBSERVE`	The event represents neither adding nor removing children from the aggregation. The observation may be incomplete: there may be children that are part of the aggregation but not observed during this event and therefore not included in the `childEPCs` or `childQuantityList` field of the `AggregationEvent`; likewise, the parent identity may not be observed or known during this event and therefore the `parentID` field be omitted from the `AggregationEvent`.
`DELETE`	The objects identified in the child list have been disaggregated from the parent during this event. This includes situations where a subset of children are removed from the aggregation, as well as when the entire aggregation is dismantled. Both `childEPCs` and `childQuantityList` field may be omitted from the `AggregationEvent`, which means that *all* children have been disaggregated. (This permits disaggregation when the event capture software does not know the identities of all the children.)

The AggregationEvent type includes fields that refer to a single "parent" (often a "containing" entity) and one or more "children" (often "contained" objects). A parent identifier is required when action is ADD or DELETE, but optional when action is OBSERVE.

Non-Normative: Explanation: A parent identifier is used when action is ADD so that there is a way of referring to the aggregation in subsequent events when action is DELETE. The parent identifier is optional when action is OBSERVE because the parent is not always known during an intermediate observation. For example, a pallet receiving process may rely on RFID tags to determine the EPCs of cases on the pallet, but there might not be an RFID tag for the pallet (or if there is one, it may be unreadable).

The AggregationEvent is intended to indicate aggregations among objects, and so the children are identified by EPCs and/or EPC classes. The parent entity, however, is not necessarily a physical or digital object separate from the aggregation itself, and so the parent is identified by an arbitrary URI, which MAY be an EPC, but MAY be another identifier drawn from a suitable private vocabulary.

Non-Normative: Explanation: In many manufacturing operations, for example, it is common to create a load several steps before an EPC for the load is assigned. In such situations, an internal tracking number (often referred to as a "license plate number," or LPN) is assigned at the time the load is created, and this is used up to the point of shipment. At the point of shipment, an SSCC code (which *is* an EPC) is assigned. In EPCIS, this would be modelled by (a) an AggregationEvent with action equal to ADD at the time the load is created, (b) an invalidation of the first association via an AggregationEvent with action equal to DELETE when the SSCC is ready to be assigned, and (c) a second AggregationEvent with action equal to ADD at the time the SSCC is assigned. During (a) and (b), the LPN would be the parent identifier (expressed in a suitable URI representation; see section 6.4), while the AggregationEvent during (c) would use the SSCC (which is a type of EPC) as the parent identifier, thereby changing the parentID.

An additional example comes from logistics operations where heterogenous sets of items are often assembled together for shipping efficiencies. Commonly, order fulfillments to retail locations from distribution centers require a wider variety of items but in smaller quantities. The items combined for transportation is a logistics unit and is assigned an SSCC as the assembly is completed. In EPCIS, this would be modelled by an AggregationEvent with action equal to ADD at the time the unit is assembled. Suppose the unit was assembled, initially, to be shipped by truck but later determined that parcel shipping was necessary to meet timing constraints of the order. In this situation, an AggregationEvent with action equal to DELETE is needed to reflect the items have been disassembled. These two events would be followed by an AggregationEvent with action equal to ADD with a new SSCC for each parcel and containing items.

An AggregationEvent has the following fields:

Field	Type	Description
eventTime recordTime eventTimeZoneOffset	(Inherited from EPCISEvent; see section 7.4.1)	

Field	Type	Description
parentID	URI	(Optional when action is OBSERVE, required otherwise) The identifier of the parent object of the aggregation. When the parent identifier is an EPC, this field SHALL contain the "pure identity" URI for the EPC as specified in [TDS].
childEPCs	List<EPC>	(Optional) An unordered list of the EPCs of contained objects identified by instance-level identifiers. See section 8.2 of the CBV. An AggregationEvent SHALL contain either a non-empty childEPCs, a non-empty childQuantityList, or both, except that both childEPCs and childQuantityList MAY be empty if action is DELETE, indicating that all children are disaggregated from the parent.
childQuantityList	List<QuantityElement>	(Optional) An unordered list of one or more QuantityElements identifying (at the class level) contained objects. See section 8.3 of the CBV. An AggregationEvent SHALL contain either a non-empty childEPCs, a non-empty childQuantityList, or both, except that both childEPCs and childQuantityList MAY be empty if action is DELETE, indicating that all children are disaggregated from the parent.
action	Action	How this event relates to the lifecycle of the aggregation named in this event. See above for more detail.
bizStep	BusinessStepID	(Optional) The business step of which this event was a part.
disposition	DispositionID	(Optional) The business condition of the objects, in the parentID, childEPCs and childQuantityList, presumed to hold true until contradicted by a subsequent event.
readPoint	ReadPointID	(Optional) The read point at which the event took place.
bizLocation	BusinessLocationID	(Optional) The business location where the aggregated objects, in the parentID, childEPCs and childQuantityList, may be found, until contradicted by a subsequent event.
bizTransactionList	Unordered list of zero or more BusinessTransaction instances	(Optional) An unordered list of business transactions that define the context of this event.
sourceList	List<Source>	(Optional) An unordered list of Source elements (section 7.3.6.4) that provide context about the originating endpoint of a business transfer of which this event is a part.
destinationList	List<Destination>	(Optional) An unordered list of Destination elements (section 7.3.6.4) that provide context about the terminating endpoint of a business transfer of which this event is a part.

Field	Type	Description
sensorElementList	List<sensorElement>	(Optional) An unordered list of one or more sensorElements (section 7.3.7).

Note that in the XML binding (section 9.3), childQuantityList, sourceList, destinationList and sensorElementList appear in the standard extension area, to maintain backward-compatibility with EPCIS 1.0, 1.1 and 1.2.

Retrospective semantics:

- An event described by bizStep (and any other fields) took place involving containing entity parentID and the contained objects in childEPCs and childQuantityList, at eventTime and location readPoint.
- (If action is ADD) The objects identified in childEPCs and childQuantityList were aggregated to containing entity parentID.
- (If action is DELETE and childEPCs or childQuantityList is non-empty) The objects identified in childEPCs and childQuantityList were disaggregated from parentID.
- (If action is DELETE and both childEPCs and childQuantityList are empty) All contained objects have been disaggregated from containing entity parentID.
- (If action is ADD and a non-empty bizTransactionList is specified) An aggregation exists between the business transactions enumerated in bizTransactionList, the objects identified in childEPCs and childQuantityList, and containing entity parentID.
- (If action is OBSERVE and a non-empty bizTransactionList is specified) This event took place within the context of the business transactions enumerated in bizTransactionList.
- (If action is DELETE and a non-empty bizTransactionList is specified) This event took place within the context of the business transactions enumerated in bizTransactionList.
- (If sourceList is non-empty) This event took place within the context of a business transfer whose originating endpoint is described by the sources enumerated in sourceList.
- (If destinationList is non-empty) This event took place within the context of a business transfer whose terminating endpoint is described by the destinations enumerated in destinationList.

Prospective semantics:

- (If action is ADD) An aggregation exists between containing entity parentID and the contained objects in childEPCs and childQuantityList.
- (If action is DELETE and childEPCs or childQuantityList is non-empty) An aggregation no longer exists between containing entity parentID and the contained objects identified in childEPCs and childQuantityList.
- (If action is DELETE and both childEPCs and childQuantityList are empty) An aggregation no longer exists between containing entity parentID and any contained objects.
- (If disposition is specified) The business condition of the objects aggregated with the objects identified in parentID, childEPCs, and childQuantityList is as described by disposition.
- (If disposition is omitted) The business condition of the objects aggregated with the objects in parentID, childEPCs, and childQuantityList is unchanged.

- (If `bizLocation` is specified) The objects aggregated with the objects in `parentID`, `childEPCs`, and `childQuantityList` are at business location `bizLocation`.
- (If `bizLocation` is omitted) The business location of the objects aggregated with the objects in `parentID`, `childEPCs`, and `childQuantityList` might be unknown, unless the business location can be inferred from previous events.
- (If `action` is `ADD` and a non-empty `bizTransactionList` is specified) An aggregation exists between the business transactions enumerated in `bizTransactionList`, the objects in `childEPCs` and `childQuantityList`, and containing entity `parentID` (if specified).

Non-Normative: Explanation: In the case where action is ADD and a non-empty `bizTransactionList` is specified, the semantic effect is equivalent to having an AggregationEvent with no `bizTransactionList` together with a TransactionEvent having the `bizTransactionList` and all same field values as the AggregationEvent. Note, however, that an AggregationEvent with a non-empty `bizTransactionList` does not cause a TransactionEvent to be returned from a query.

Non-Normative: Note: Many semantically invalid situations can be expressed with incorrect use of aggregation. For example, the same objects may be given multiple parents during the same time period by distinct ADD operations without an intervening Delete. Similarly, an object can be specified to be a child of its grand-parent or even of itself. A non-existent aggregation may be DELETED. These situations cannot be detected syntactically and in general an individual EPCIS repository may not have sufficient information to detect them. Thus, this specification does not address these error conditions.

7.4.4 TransactionEvent (subclass of EPCISEvent)

The event type `TransactionEvent` describes the association or disassociation of physical or digital objects to one or more business transactions. While other event types have an optional `bizTransactionList` field that may be used to provide context for an event, the `TransactionEvent` is used to declare in an unequivocal way that certain objects have been associated or disassociated with one or more business transactions as part of the event.

The `action` field of a `TransactionEvent` describes the event's relationship to the lifecycle of the transaction. Specifically:

`Action` value	Meaning
`ADD`	The objects identified in the event have been associated to the business transaction(s) during this event. This includes situations where the transaction(s) is created for the first time, as well as when new objects are added to an existing transaction(s).

Action value	Meaning
OBSERVE	The objects named in the event have been confirmed as continuing to be associated to the business transaction(s) during this event. Explanation (non-normative): A TransactionEvent with action OBSERVE is quite similar to an ObjectEvent that includes a non-empty bizTransactionList field. When an end user group agrees to use both kinds of events, the group should clearly define when each should be used. An example where a TransactionEvent with action OBSERVE might be appropriate is an international shipment with transaction ID xxx moving through a port, and there's a desire to record the EPCs that were observed at that point in handling that transaction. Subsequent queries will concentrate on querying the transaction ID to find the EPCs, not on the EPCs to find the transaction ID.
DELETE	The objects named in the event have been disassociated from the business transaction(s) during this event. This includes situations where a subset of objects are disassociated from the business transaction(s), as well as when the entire business transaction(s) has ended. As a convenience, both the list of EPCs and QuantityElements may be omitted from the TransactionEvent, which means that *all* objects have been disassociated.

A TransactionEvent has the following fields:

Field	Type	Description
eventTime recordTime eventTimeZoneOffset	(Inherited from EPCISEvent; see section 7.4.1)	
bizTransactionList	Unordered list of one or more BusinessTransaction instances	The business transaction(s).
parentID	URI	(Optional) The identifier of the parent object of objects listed in epcList and quantityList, if these fields are present. When the parent identifier is an EPC, this field SHALL contain the "pure identity" URI for the EPC as specified in [TDS]. See also the note following the table.
epcList	List<EPC>	(Optional) An unordered list of the EPCs of the objects identified by instance-level identifiers associated with the business transaction. See section 8.2 of the CBV. A TransactionEvent SHALL contain either a non-empty epcList, a non-empty quantityList, or both, except that both epcList and quantityList MAY be empty if action is DELETE, indicating that all the objects are disassociated from the business transaction(s).

Field	Type	Description
quantityList	List<QuantityElement>	(Optional) An unordered list of one or more QuantityElements identifying objects (at the class level) to which the event pertained. A TransactionEvent SHALL contain either a non-empty epcList, a non-empty quantityList, or both, except that both epcList and quantityList MAY be empty if action is DELETE, indicating that all the objects are disassociated from the business transaction(s).
action	Action	How this event relates to the lifecycle of the business transaction named in this event. See above for more detail.
bizStep	BusinessStepID	(Optional) The business step of which this event was a part.
disposition	DispositionID	(Optional) The business condition of the objects, in the epcList and quantityList, presumed to hold true until contradicted by a subsequent event.
readPoint	ReadPointID	(Optional) The read point at which the event took place.
bizLocation	BusinessLocationID	(Optional) The business location where the objects associated with the containing and contained objects may be found, until contradicted by a subsequent event.
sourceList	List<Source>	(Optional) An unordered list of Source elements (section 7.3.6.4) that provide context about the originating endpoint of a business transfer of which this event is a part.
destinationList	List<Destination>	(Optional) An unordered list of Destination elements (section 7.3.6.4) that provide context about the terminating endpoint of a business transfer of which this event is a part.
sensorElementList	List<sensorElement>	(Optional) An unordered list of one or more sensorElements (section 7.3.7).

Note that in the XML binding (section 9.3), quantityList, sourceList, destinationList and sensorElementList appear in the standard extension area, to maintain backward-compatibility with EPCIS 1.0, 1.1 and 1.2.

Non-Normative: Explanation: The use of the field name parentID in both TransactionEvent and AggregationEvent (section 7.2.10) does not indicate a similarity in function or semantics. In general, a TransactionEvent carries the same object identification information as an ObjectEvent, that is, a list of EPCs and/or QuantityElements. All the other information fields (bizTransactionList, bizStep, bizLocation, etc) apply equally and uniformly to all objects specified, whether or not the objects are specified in just

the epcList and quantityList field or if the optional parentID field is also supplied.

Non-Normative: The TransactionEvent provides a way to describe the association or disassociation of business transactions to objects. The parentID field in the TransactionEvent highlights a specific EPC or other identifier as the preferred or primary object but does not imply a physical relationship of any kind, nor is any kind of nesting or inheritance implied by the TransactionEvent itself. Only AggregationEvent instances describe actual parent-child relationships and nestable parent-child relationships. This can be seen by comparing the semantics of AggregationEvent in section 7.2.10 with the semantics of TransactionEvent below.

Retrospective semantics:

- An event described by bizStep (and any other fields) took place involving the business transactions enumerated in bizTransactionList, the objects in epcList and quantityList, and containing entity parentID (if specified), at eventTime and location readPoint.
- (If action is ADD) The objects in epcList and quantityList and containing entity parentID (if specified) were associated to the business transactions enumerated in bizTransactionList.
- (If action is DELETE and epcList or quantityList is non-empty) The objects in epcList, quantityList, and containing entity parentID (if specified) were disassociated from the business transactions enumerated in bizTransactionList.
- (If action is DELETE, both epcList and quantityList are empty, and parentID is omitted) All objects have been disassociated from the business transactions enumerated in bizTransactionList.
- (If sourceList is non-empty) This event took place within the context of a business transfer whose originating endpoint is described by the sources enumerated in sourceList.
- (If destinationList is non-empty) This event took place within the context of a business transfer whose terminating endpoint is described by the destinations enumerated in destinationList.

Prospective semantics:

- (If action is ADD) An association exists between the business transactions enumerated in bizTransactionList, the objects in epcList and quantityList, and containing entity parentID (if specified).
- (If action is DELETE and epcList or quantityList is non-empty) An association no longer exists between the business transactions enumerated in bizTransactionList, the objects in epcList and quantityList, and containing entity parentID (if specified).
- (If action is DELETE, both epcList and quantityList are empty, and parentID is omitted) An association no longer exists between the business transactions enumerated in bizTransactionList and any objects.
- (If disposition is specified) The business condition of the objects associated with the objects in epcList and quantityList and containing entity parentID (if specified) is as described by disposition.
- (If disposition is omitted) The business condition of the objects associated with the objects in epcList and quantityList and containing entity parentID (if specified) is unchanged.

- (If `bizLocation` is specified) The objects associated with the objects in `epcList`, `quantityList`, and containing entity `parentID` (if specified) are at business location `bizLocation`.
- (If `bizLocation` is omitted) The business location of the objects associated with the objects in `epcList` and `quantityList` and containing entity `parentID` (if specified) might be unknown, unless the business location can be inferred from previous events.

7.4.5 TransformationEvent (subclass of EPCISEvent)

A `TransformationEvent` captures information about an event in which one or more physical or digital objects identified by instance-level (EPC) or class-level (EPC Class) identifiers are fully or partially consumed as inputs and one or more objects identified by instance-level (EPC) or class-level (EPC Class) identifiers are produced as outputs. The `TransformationEvent` captures the relationship between the inputs and the outputs, such that any of the inputs may have contributed in some way to each of the outputs.

Some transformation business processes take place over a long period of time, and so it is more appropriate to represent them as a series of EPCIS events. A `transformationID` may be included in two or more `TransformationEvents` to link them together. When events share an identical `transformationID`, the meaning is that the inputs to *any* of those events may have contributed in some way to each of the outputs in *any* of those same events.

Fields:

Field	Type	Description

`eventTime`
`recordTime`
`eventTimeZoneOffset`
	(Inherited from `EPCISEvent`; see section 7.4.1)	

`inputEPCList`	`List<EPC>`	(Optional) An unordered list of one or more EPCs identifying (at the instance level) objects that were inputs to the transformation..
`inputQuantityList`	`List<QuantityElement>`	(Optional) An unordered list of one or more `QuantityElement`s identifying (at the class level) objects that were inputs to the transformation.
`outputEPCList`	`List<EPC>`	(Optional) An unordered list of one or more EPCs naming (at the instance level) objects that were outputs from the transformation.
`outputQuantityList`	`List<QuantityElement>`	(Optional) An unordered list of one or more `QuantityElement`s identifying (at the class level) objects that were outputs from the transformation.
`transformationID`	`TransformationID`	(Optional) A unique identifier that links this event to other TransformationEvents having an identical value of `transformationID`. When specified, all inputs to all events sharing the same value of the `transformationID` may contribute to all outputs of all events sharing that value of `transformationID`. If `transformationID` is omitted, then the inputs of this event may contribute to the outputs of this event, but the inputs and outputs of other events are not connected to this one.
`bizStep`	`BusinessStepID`	(Optional) The business step of which this event was a part.

Field	Type	Description
`disposition`	`DispositionID`	(Optional) The business condition of the objects, in the `outputEPCList` and `outputQuantityList`, presumed to hold true until contradicted by a subsequent event.
`persistentDisposition`	`PersistentDisposition`	(Optional) One or more business conditions of the objects, in the outputEPCList and outputQuantityList. Each persistentDisposition is explicitly `set` and `unset` independently of other `persistentDisposition` values. The `set` field within `persistentDisposition` may specify a list of `persistentDisposition` URI values to be set. The `unset` field within `persistentDisposition` may specify a list of `persistentDisposition` URI values to be unset (revoked).
`set`	`DispositionID`	(Optional, multivalued) If used in `PersistentDisposition`, specifies `Disposition` (`cbv:Disp`) values to be **`set` "persistently"**, i.e., **until they are explicitly `unset`**.
`unset`	`DispositionID`	(Optional, multivalued) If used in `PersistentDisposition`, specifies `Disposition` (`cbv:Disp`) values to be **`unset` "persistently"**, i.e., **until they are explicitly `set`**.
`readPoint`	`ReadPointID`	(Optional) The read point at which the event took place.
`bizLocation`	`BusinessLocationID`	(Optional) The business location where the output objects of this event may be found, until contradicted by a subsequent event.
`bizTransactionList`	Unordered list of zero or more `BusinessTransaction` instances	(Optional) An unordered list of business transactions that define the context of this event.
`sourceList`	`List<Source>`	(Optional) An unordered list of `Source` elements (section 7.3.6.4) that provide context about the originating endpoint of a business transfer of which this event is a part.
`destinationList`	`List<Destination>`	(Optional) An unordered list of `Destination` elements (section 7.3.6.4) that provide context about the terminating endpoint of a business transfer of which this event is a part.

Field	Type	Description
ilmd	ILMD	(Optional) Instance/Lot master data (section 7.3.8) that describes the output objects created during this event.
sensorElementList	List<sensorElement>	(Optional) An unordered list of one or more sensorElements (section 7.3.7).

Note that in the XML binding (section 9.3), sensorElementList appears in the standard extension area, to maintain backward-compatibility with EPCIS 1.2.

If transformationID is omitted, then a TransformationEvent SHALL include at least one input (i.e., at least one of inputEPCList and inputQuantityList are non-empty) AND at least one output (i.e., at least one of outputEPCList and outputQuantityList are non-empty). If transformationID is included, then a TransformationEvent SHALL include at least one input OR at least one output (or both). The latter provides for the possibility that in a transformation described by several events linked by a common transformationID, any one event might only add inputs or extract outputs.

Retrospective semantics:

- A transformation described by bizStep (and any other fields) took place with input objects identified by inputEPCList and inputQuantityList and output objects identified by outputEPCList and outputQuantityList, at eventTime at location readPoint.
- This event took place within the context of the business transactions enumerated in bizTransactionList.
- (If transformationID is omitted) Any of the input objects identified by inputEPCList and inputQuantityList of this event may have contributed to each of the output objects identified by outputEPCList and outputQuantityList of this event.
- (If transformationID is included) Any of the input objects identified by inputEPCList and inputQuantityList of this event, together with the input objects identified by inputEPCList and inputQuantityList of other events having the same value of transformationID, may have contributed to each of the output objects identified by outputEPCList and outputQuantityList of this event, as well as to each of the output objects identified by outputEPCList and outputQuantityList of other events having the same value of transformationID.
- (If sourceList is non-empty) This event took place within the context of a business transfer whose originating endpoint is described by the sources enumerated in sourceList.
- (If destinationList is non-empty) This event took place within the context of a business transfer whose terminating endpoint is described by the destinations enumerated in destinationList.

Prospective semantics:

- The objects identified by the instance-level identifiers in outputEPCList may appear in subsequent events.
- The objects identified by the class-level identifiers in outputQuantityList may appear in subsequent events.
- (If disposition is specified) The business condition of the objects identified by outputEPCList and outputQuantityList is as described by disposition.
- (If disposition is omitted) The business condition of the objects associated with identified by outputEPCList and outputQuantityList is unknown.

- (If a specific `persistentDisposition` is specified as `set`) The persistent business condition(s) of the objects identified by `outputEPCList` and `outputQuantityList` is as set by `persistentDisposition`.
- (If `persistentDisposition` is omitted) The persistent business condition(s) of the objects identified by `outputEPCList` and `outputQuantityList` as previously `set` or `unset` by `persistentDisposition is unchanged`.
- (If a specific `persistentDisposition` value is specified as `unset`) The specific persistent `business` condition of the objects identified by `outputEPCList` and `outputQuantityList` is unset (i.e., revoked).
- (If `bizLocation` is specified) `The` objects identified by `outputEPCList` and `outputQuantityList` are at business location `bizLocation`.
- (If `bizLocation` is omitted) The business location of the objects identified by `outputEPCList` and `outputQuantityList` might be unknown, unless the business location can be inferred from previous events.
- (If `ilmd` is non-empty) The objects identified by `outputEPCList` and `outputQuantityList` are described by the attributes in `ilmd`.

7.4.6 AssociationEvent (subclass of EPCISEvent)

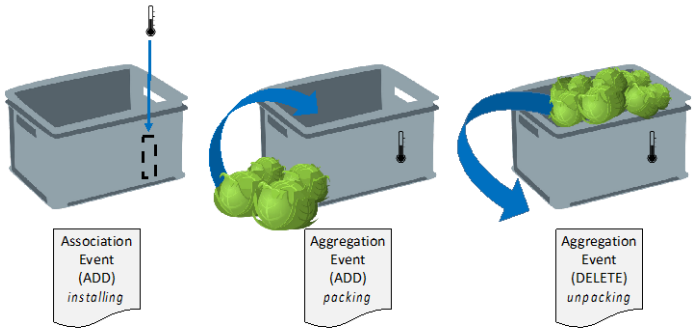
The event type `AssociationEvent` describes the association or disassociation of one or several physical objects with a parent object or a specific physical location.

Like the `AggregationEvent`, the `AssociationEvent` is also used to capture associations where there is a strong physical relationship between the containing and the contained objects such that they will all occupy the same location at the same time, until such time as they are disaggregated. However, the `AggregationEvent` does not allow for associations of objects with physical locations; if `action` is DELETE while omitting the `childEPC` and `childQuantityList` field, all contained children are disaggregated from the containing parent.

Because there are situations in which associations are more permanent, i.e. beyond the physical flow of goods (e.g. packing/unpacking and loading/unloading), an `AssociationEvent` SHOULD be used: (a) when objects need to be associated with a physical location or (b) when the parent object could also be subject to other, more temporary associations (i.e. captured with `AggregationEvent`s).

Non-Normative: Explanation on why the `AssociationEvent` is required:

Figure 7-6 Association and Aggregation with returnable transport units (RTIs)



Assume that an `AggregationEvent` is used to capture the construction of a wagon or the installation of sensors into a reusable conveyance (RTI). Later on, the identifiers of these units also appear in other `AggregationEvent`s, documenting the physical flow of goods (e.g. pallets are loaded onto a lorry, maritime containers are loaded on a ship, or vegetables are packed into a plastic transport box). If an organisation captures an `AggregationEvent` with `action` DELETE while not indicating any of the aggregated children, it would mean that ALL children ever aggregated to that parent ID are no longer aggregated to it – i.e. not just the pallets, the maritime containers or the vegetables, but also all assemblies, screws, tyres, sensors, etc. used to assemble those parent units. Use of an `AssociationEvent` for construction/installation processes removes the ambiguity, allowing an accessing application to infer which children are inherent parts and which of them are only temporary parts of the association with the parent object.

Figure 7-7 Association and Aggregation with containers

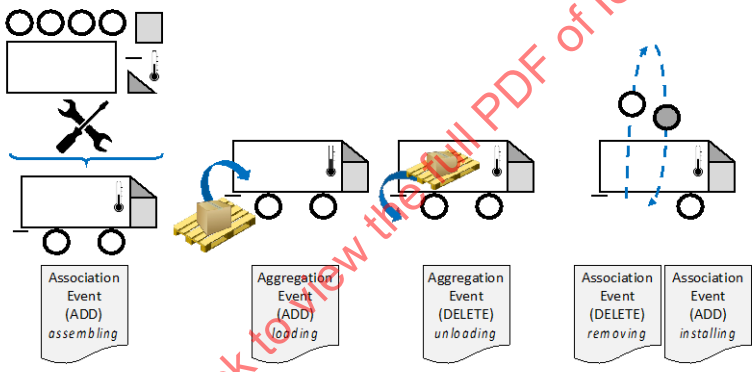
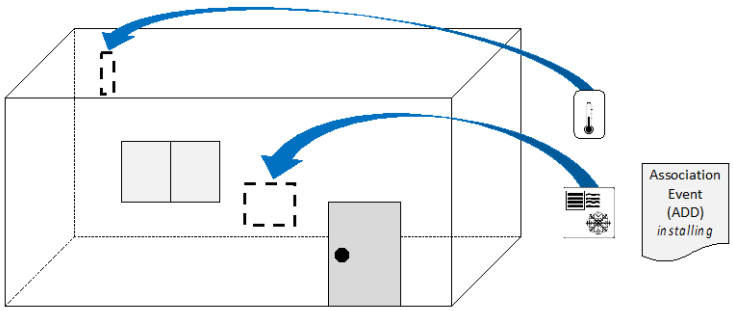


Figure 7-8 Association and Aggregation in a room



The three examples above illustrate the respective applicability of the `AggregationEvent` and the `AssociationEvent` in common business process flows:

The `AssociationEvent` SHALL be the only event type in which it is permissible to populate the `parentID` field with a location ID (for instance, a GLN with or without extension).

The `action` field of an `AssociationEvent` describes the event's relationship to the lifecycle of the association. Specifically:

Action value	Meaning
`ADD`	The objects identified in the child list have been associated to the parent during this event. This includes situations where the association is created for the first time, as well as when new children are added to an existing association.
`OBSERVE`	The event represents neither adding nor removing children from the association. The observation may be incomplete: there may be children that are part of the association but not observed during this event and therefore not included in the `childEPCs` or `childQuantityList` field of the `AssociationEvent`; likewise, the parent identity may not be observed or known during this event and therefore the `parentID` field be omitted from the `AssociationEvent`.
`DELETE`	The objects identified in the child list have been disassociated from the parent during this event. This includes situations where a subset of children are removed from the association, as well as when the entire association is dismantled. Both `childEPCs` and `childQuantityList` field may be omitted from the `AssociationEvent`, which means that *all* children have been disassociated. (This permits disassociation when the event capture software does not know the identities of all the children.)

The `AssociationEvent` type includes fields that refer to a single "parent" (often a "containing" entity) and one or more "children" (often "contained" objects). A parent identifier is required when `action` is `ADD` or `DELETE`, but optional when `action` is `OBSERVE`.

Non-Normative: Explanation: A parent identifier is used when `action` is `ADD` so that there is a way of referring to the association in subsequent events when `action` is `DELETE`. The parent identifier is optional when `action` is `OBSERVE` because the parent is not always known during an intermediate observation.

An `AssociationEvent` has the following fields:

Field	Type	Description

`eventTime`
`recordTime`
`eventTimeZoneOffset`
	(Inherited from `EPCISEvent`; see section 7.3.)	

`parentID`	`URI`	The identifier of the parent object or parent location of the association. When the parent identifier is an EPC, this field SHALL contain the "pure identity" URI for the EPC as specified in [TDS].

Field	Type	Description
childEPCs	List<EPC>	(Optional) An unordered list of the EPCs of contained objects identified by instance-level identifiers. An AssociationEvent SHALL contain either a non-empty childEPCs, a non-empty childQuantityList, or both, except that both childEPCs and childQuantityList MAY be empty if action is DELETE, indicating that all children are disassociated from the parent.
childQuantityList	List<QuantityElement>	(Optional) An unordered list of one or more QuantityElements identifying (at the class level) contained objects. See section 7.3.3.1. An AssociationEvent SHALL contain either a non-empty childEPCs, a non-empty childQuantityList, or both, except that both childEPCs and childQuantityList MAY be empty if action is DELETE, indicating that all children are disaggregated from the parent.
action	Action	How this event relates to the lifecycle of the association named in this event. See above for more detail.
bizStep	BusinessStepID	(Optional) The business step of which this event was a part.
disposition	DispositionID	(Optional) The business condition of the object, in the parentID, childEPCs and childQuantityList, presumed to hold true until contradicted by a subsequent event.
readPoint	ReadPointID	(Optional) The read point at which the event took place.
bizLocation	BusinessLocationID	(Optional) The business location where the objects associated with the containing and contained EPCs may be found, until contradicted by a subsequent event.
bizTransactionList	Unordered list of zero or more BusinessTransaction instances	(Optional) An unordered list of business transactions that define the context of this event.

Field	Type	Description
sourceList	List<Source>	(Optional) An unordered list of Source elements (section 7.3.6.4) that provide context about the originating endpoint of a business transfer of which this event is a part.
destinationList	List<Destination>	(Optional) An unordered list of Destination elements (section 7.3.6.4) that provide context about the terminating endpoint of a business transfer of which this event is a part.
sensorElementList	List<sensorElement>	(Optional) An unordered list of one or more sensorElements (section 7.3.7.1).

Non-Normative: The parentID of a AssociationEvent SHOULD have been captured in a prior commissioning event, even if the object is simply the planned result of associating / assembling the child components, even if they are not yet mounted/installed.

Retrospective semantics:

- An event described by bizStep (and any other fields) took place involving containing entity parentID and the contained objects in childEPCs and childQuantityList, at eventTime and location readPoint.
- (If action is ADD) The objects identified in childEPCs and childQuantityList were associated to containing entity parentID.
- (If action is DELETE and childEPCs or childQuantityList is non-empty) The objects identified in childEPCs and childQuantityList were disassociated from parentID.
- (If action is DELETE and both childEPCs and childQuantityList are empty) All contained objects have been disassociated from containing entity parentID.
- (If action is ADD and a non-empty bizTransactionList is specified) An association exists between the business transactions enumerated in bizTransactionList, the objects identified in childEPCs and childQuantityList, and containing entity parentID.
- (If action is OBSERVE and a non-empty bizTransactionList is specified) This event took place within the context of the business transactions enumerated in bizTransactionList.
- (If action is DELETE and a non-empty bizTransactionList is specified) This event took place within the context of the business transactions enumerated in bizTransactionList.
- (If sourceList is non-empty) This event took place within the context of a business transfer whose originating endpoint is described by the sources enumerated in sourceList.
- (If destinationList is non-empty) This event took place within the context of a business transfer whose terminating endpoint is described by the destinations enumerated in destinationList.
- (If sensorElementList is non-empty) This event took place in the context of the sensor observation specified in the sensorElementList at time or during startTime and endTime (or at eventTime when time, startTime and

endTime are omitted). All values pertain to the objects identified by epcList and quantityList.

Prospective semantics:

- (If action is ADD) An association exists between containing entity parentID and the contained objects in childEPCs and childQuantityList.
- (If action is DELETE and childEPCs or childQuantityList is non-empty) An association no longer exists between containing entity parentID and the contained objects identified in childEPCs and childQuantityList.
- (If action is DELETE and both childEPCs and childQuantityList are empty) An association no longer exists between containing entity parentID and any contained objects.
- (If disposition is specified) The business condition of the objects associated with the objects identified in parentID, childEPCs, and childQuantityList is as described by disposition.
- (If disposition is omitted) The business condition of the objects associated with the objects in parentID, childEPCs, and childQuantityList is unchanged.
- (If bizLocation is specified) The objects associated with the objects in parentID, childEPCs, and childQuantityList are at business location bizLocation.
- (If bizLocation is omitted) The business location of the objects associated with the objects in parentID, childEPCs, and childQuantityList might be unknown, unless the business location can be inferred from previous events.
- (If action is ADD and a non-empty bizTransactionList is specified) An association exists between the business transactions enumerated in bizTransactionList, the objects in childEPCs and childQuantityList, and containing entity parentID (if specified).

Non-Normative: Explanation: In the case where action is ADD and a non-empty bizTransactionList is specified, the semantic effect is equivalent to having an AssociationEvent with no bizTransactionList together with a TransactionEvent having the bizTransactionList and all same field values as the AssociationEvent. Note, however, that an AssociationEvent with a non-empty bizTransactionList does not cause a TransactionEvent to be returned from a query.

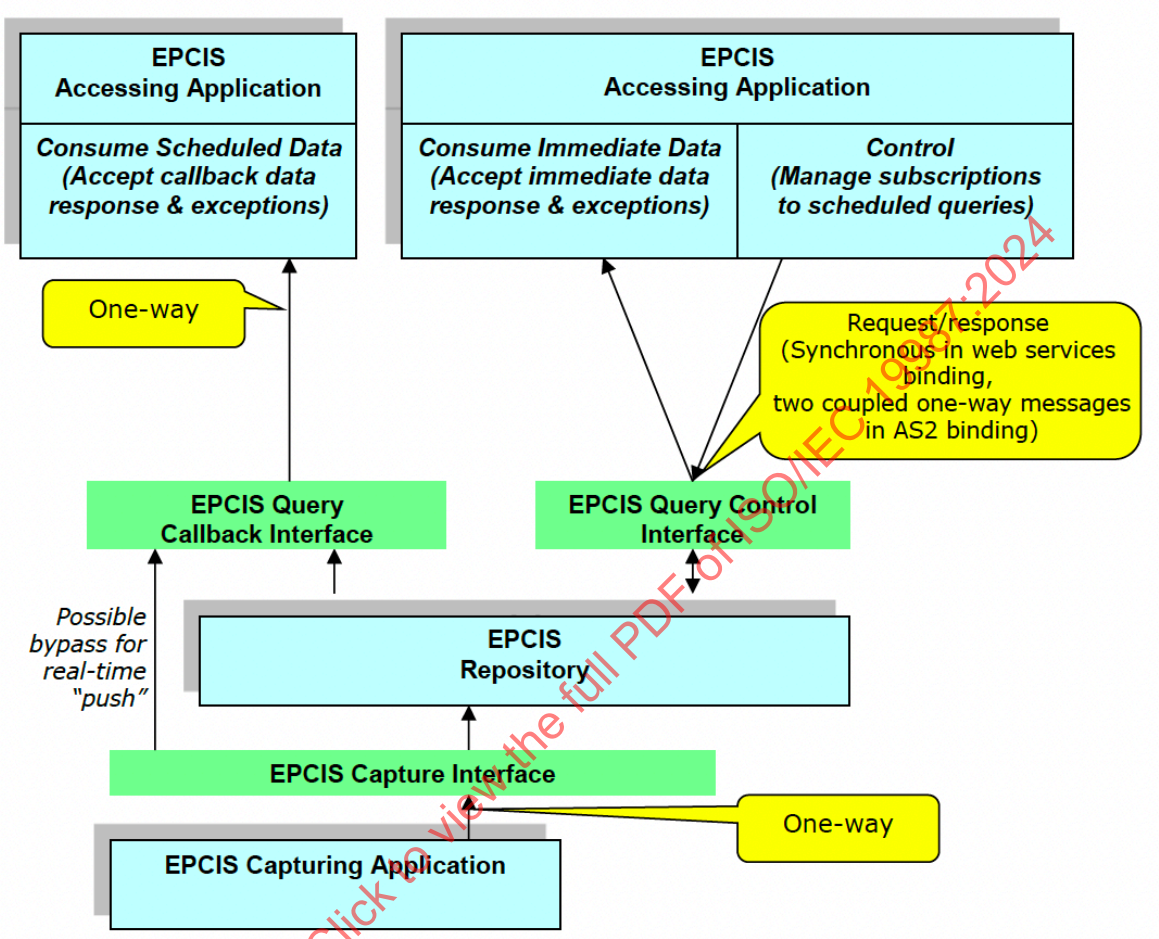
Non-Normative: Note: Many semantically invalid situations can be expressed with incorrect use of association. For example, the same objects may be given multiple parents during the same time period by distinct ADD operations without an intervening DELETE. Similarly, an object can be specified to be a child of its grand-parent or even of itself. A non-existent association may be deleted. These situations cannot be detected syntactically and in general an individual EPCIS repository may not have sufficient information to detect them. This specification does not address such situations.

8 Service Layer

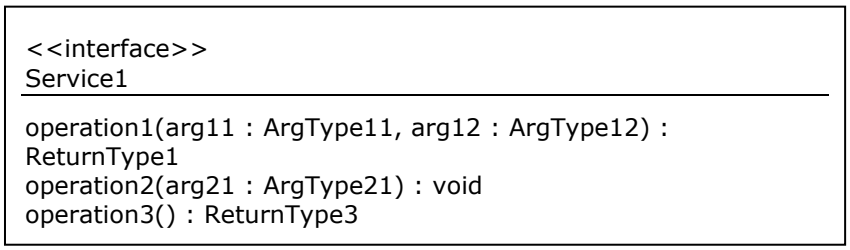
This section includes normative specifications of modules in the Service Layer. Together, these modules define three interfaces: the EPCIS Capture Interface, the EPCIS Query Control Interface, and the EPCIS Query Callback Interface. (The latter two interfaces are referred to collectively as the EPCIS Query Interfaces.)

The diagram below illustrates the relationship between these interfaces, expanding upon the diagram in section 2 (this diagram is non-normative).

Figure 8-1 EPCIS Service Layer



In the subsections below, services are specified using UML class diagram notation. UML class diagrams used for this purpose may contain interfaces having operations, but not fields or associations. Here is an example:



This diagram shows a service definition for `Service1`, which provides three operations. `Operation1` takes two arguments, `arg11` and `arg12`, having types `ArgType11` and `ArgType12`, respectively, and returns a value of type `ReturnType1`. `Operation2` takes one argument but does not return a result. `Operation3` does not take any arguments but returns a value of type `ReturnType3`.

Within the UML descriptions, the notation `<<extension point>>` identifies a place where implementations SHALL provide for extensibility through the addition of new operations. Extensibility mechanisms SHALL provide for both proprietary extensions by vendors of EPCIS-compliant products, and for extensions defined by GS1 through future versions of this specification or through new specifications.

In the case of the standard WSDL bindings, the extension points are implemented simply by permitting the addition of additional operations.

8.1 Core capture operations module

The Core Capture Operations Module provides operations by which core events may be delivered from an EPCIS Capture Application. Within this section, the word "client" refers to an EPCIS Capture Application and "EPCIS Service" refers to a system that implements the EPCIS Capture Interface.

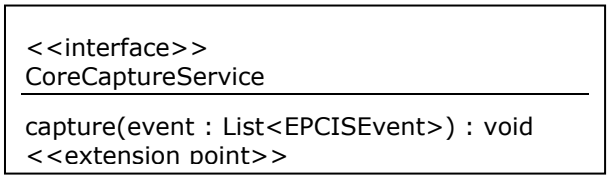
8.1.1 Authentication and authorisation

Some bindings of the EPCIS Capture Interface provide a means for the EPCIS Service to authenticate the client's identity, for the client to authenticate the EPCIS Service's identity, or both. The specification of the means to authenticate is included in the specification of each binding. If the EPCIS Service authenticates the identity of the client, an implementation MAY use the client identity to make authorisation decisions as described below. Moreover, an implementation MAY record the client identity with the captured data, for use in subsequent authorisation decisions by the system implementing the EPCIS Query Interfaces, as described in section 8.2.2.

Because of the simplicity of the EPCIS Capture Interface, the authorisation provisions are very simple to state: namely, an implementation MAY use the authenticated client identity to decide whether a capture operation is permitted or not.

Non-Normative: Explanation: It is expected that trading partners will always use bindings that provide for client identity authentication or mutual authentication when using EPCIS interfaces to share data across organisational boundaries. The bindings that do not offer authentication are expected to be used only within a single organisation in situations where authentication is not required to meet internal security requirements.

8.1.2 Capture service



The capture interface contains only a single method, `capture`, which takes a single argument; if events are successfully accepted, the capture interface responds with `202 Accepted` and a `captureID`. Implementations of the EPCIS Capture Interface SHALL accept each element of the argument list that is a valid `EPCISEvent` or subtype thereof according to this specification. Implementations MAY accept other types of events through vendor extension. The simplicity of this interface admits a wide variety of bindings, including simple message-queue type bindings.

Non-Normative: Explanation: "Message-queue type bindings" means the following. Enterprises commonly use "message bus" technology for interconnection of different distributed system components. A message bus provides a reliable channel for in-order delivery of messages from a sender to a receiver. (The relationship between sender and receiver may be point-to-point (a message "queue") or one-to-many via a publish/subscribe mechanism (a message "topic").) A "message-queue type binding" of the EPCIS Capture Interface would simply be the designation of a particular message bus channel for the purpose of delivering EPCIS events from an EPCIS Capture Application to an EPCIS Repository, or to an EPCIS Accessing Application by way of the EPCIS Query Callback Interface. Each message would have a payload containing one or more EPCIS events (serialised through some binding at the Data Definition Layer; e.g., an XML binding). In such a binding, therefore, each transmission/delivery of a message corresponds to a single "capture" operation.

The `capture` operation records one or more EPCIS events, of any type.

Arguments:

Argument	Type	Description
`event`	List of `EPCISEvent`	The event(s) to capture. All relevant information such as the event time, EPCs, etc., are contained within each event. Exception: the `recordTime` MAY be omitted. Whether the `recordTime` is omitted or not in the input, following the capture operation the `recordTime` of the event as recorded by the EPCIS Repository or EPCIS Accessing Application is the time of capture. Note that the optional `eventID` is not treated like `recordTime`; like any other EPCIS field, `eventID` shall be captured without modification by the capture interface unless the `eventID` field is empty, in which case a capture interface MAY add an `eventID`. Explanation (non-normative): this treatment of `recordTime` is necessary for standing queries to be processed properly. See section 8.2.5.2.

Return value:

(none)

The concrete bindings of the EPCIS Capture Interface in section 11 use the EPCIS document structure defined in section 9.5 to carry the list of EPCIS events to be captured. An EPCIS document may contain master data in the document header. An implementation of the EPCIS Capture Interface conforming to this standard MAY choose to record that master data or MAY choose to ignore it – the disposition of

master data received through the EPCIS Capture Interface is not specified by the EPCIS standard.

On the other hand, any instance/lot master data carried in the ILMD section of an event SHALL be captured as part of the event, as is true for any data element within an EPCIS event. It SHALL be possible to query such master data by using the query parameters of the SimpleEventQuery specified in section 8.2.7.1.

An implementation of the capture interface SHALL either capture all events specified in a given capture operation or fail to capture all events in that operation. That is, an implementation SHALL NOT have the possibility of partial success where some events in the list are captured and others are not.

The reasons why a capture operation fails are implementation-specific. Examples of possible reasons a failure may occur include:

- The input to the capture operation is not well formed or does not conform to the syntactic requirements of the concrete binding being used, including schema-validity for concrete bindings that use the XML schemas defined in section 9.
- The client is not authorized to perform the capture operation.
- Implementation-specific limits regarding the number of events in a single capture operation, the total number of events stored, the frequency of capture, etc., are exceeded.
- Implementation-specific rules regarding the content of events, either in isolation or with reference to previously captured events, are violated. Note that such rules may be appropriate in a closed system where the use of EPCIS is governed by a specific application standard, but may not be appropriate in an open system designed to handle any EPCIS data. Rules of this kind may limit interoperability if they are too narrow.
- A temporary failure, such as the temporary unavailability of a server or network.

8.2 Core Query operations module

The Core Query Operations Module provides two interfaces, called the EPCIS Query Control Interface and the EPCIS Query Callback Interface, by which EPCIS data can be retrieved by an EPCIS Accessing Application. The EPCIS Query Control Interface defines a means for EPCIS Accessing Applications and trading partners to obtain EPCIS data subsequent to capture from any source, typically by interacting with an EPCIS Repository. It provides a means for an EPCIS Accessing Application to retrieve data on-demand, and also enter subscriptions for standing queries. Results of standing queries are delivered to EPCIS Accessing Applications via the EPCIS Query Callback Interface. Within this section, the word "client" refers to an EPCIS Accessing Application and "EPCIS Service" refers to a system that implements the EPCIS Query Control Interface, and in addition delivers information to a client via the EPCIS Query Callback Interface.

8.2.1 Authentication

Some bindings of the EPCIS Query Control Interface provide a means for the EPCIS Service to authenticate the client's identity, for the client to authenticate the EPCIS Service's identity, or both. The specification of the means to authenticate is included in the specification of each binding. If the EPCIS Service authenticates the identity of the client, an implementation MAY use the client identity to make authorisation decisions as described in the next section.

Non-Normative: Explanation: It is expected that trading partners will always use bindings that provide for client identity authentication or mutual authentication when using EPCIS interfaces to share data across organisational boundaries. The bindings that do not offer authentication are expected to be used only within a single organisation in situations where authentication is not required to meet internal security requirements.

8.2.2 Authorisation and redaction

An EPCIS service may wish to provide access to only a subset of information, depending on the identity of the requesting client. This situation commonly arises in cross-enterprise scenarios where the requesting client belongs to a different organisation than the operator of an EPCIS service, but may also arise in intra-enterprise scenarios.

Given an EPCIS query, an EPCIS service MAY take any of the following actions in processing the query, based on the authenticated identity of the client:

- The service MAY refuse to honour the request altogether, by responding with a `SecurityException` as defined below.
- The service MAY respond with less data than requested. For example, if a client presents a query requesting all `ObjectEvent` instances within a specified time interval, the service knows of 100 matching events, the service may choose to respond with fewer than 100 events (e.g., returning only those events whose EPCs are SGTINs with a company prefix known to be assigned to the client).
- The service MAY respond with coarser grained information. In particular, when the response to a query includes a location type (as defined in section 7.3.4), the service may substitute an aggregate location in place of a primitive location.
- The service MAY hide information. For example, if a client presents a query requesting `ObjectEvent` instances, the service may choose to delete the `bizTransactionList` fields in its response. The information returned, however, SHALL be well-formed EPCIS events consistent with this specification and industry guidelines. In addition, if hiding information would otherwise result in ambiguous, or misleading information, then the entire event SHOULD be withheld. This applies whether the original information was captured through the EPCIS Capture Interface or provided by some other means. For example, given an AggregationEvent with action equal to ADD, an attempt to hide the `parentID` field would result in a non-well-formed event, because `parentID` is required when the action is ADD; in this instance, therefore, the entire event would have to be withheld.
- The service MAY limit the scope of the query to data that was originally captured by a particular client identity. This allows a single EPCIS service to be "partitioned" for use by groups of unrelated users whose data should be kept separate.

An EPCIS implementation is free to determine which if any of these actions to take in processing any query, using any means it chooses. The specification of authorisation rules is outside the scope of this specification.

Non-Normative: Explanation: Because the EPCIS specification is concerned with the query *interfaces* as opposed to any particular implementation, the EPCIS specification does not take a position as to how authorisation decisions are taken. Particular implementations of EPCIS may have arbitrarily complex business rules for authorisation. That said, the EPCIS specification may contain standard data that is needed for authorisation, whether exclusively for that purpose or not.

8.2.3 Queries for large amounts of data

Many of the query operations defined below allow a client to make a request for a potentially unlimited amount of data. For example, the response to a query that asks for all `ObjectEvent` instances within a given interval of time could conceivably return one, a thousand, a million, or a billion events depending on the time interval and how many events had been captured. This may present performance problems for service implementations.

To mitigate this problem, an EPCIS service MAY reject any request by raising a `QueryTooLarge` exception. This exception indicates that the amount of data being requested is larger than the service is willing to provide to the client. The `QueryTooLarge` exception is a hint to the client that the client might succeed by

narrowing the scope of the original query, or by presenting the query at a different time (e.g., if the service accepts or rejects queries based on the current computational load on the service).

Non-Normative: Roadmap: It is expected that future versions of this specification will provide more sophisticated ways to deal with the large query problem, such as paging, use of cursors, etc. Nothing more complicated was agreed to in this version.

8.2.4 Overly complex queries

EPCIS service implementations may wish to restrict the kinds of queries that can be processed, to avoid processing queries that will consume more resources than the service is willing to expend. For example, a query that is looking for events having a specific value in a particular event field may require more or fewer resources to process depending on whether the implementation anticipated searching on that field (e.g., depending on whether or not a database column corresponding to that field is indexed). As with queries for too much data (section 8.2.3), this may present performance problems for service implementations.

To mitigate this problem, an EPCIS service MAY reject any request by raising a `QueryTooComplex` exception. This exception indicates that structure of the query is such that the service is unwilling to carry it out for the client. Unlike the `QueryTooLarge` exception (section 8.2.3), the `QueryTooComplex` indicates that merely narrowing the scope of the query (e.g., by asking for one week's worth of events instead of one month's) is unlikely to make the query succeed.

A particular query language may specify conditions under which an EPCIS service is not permitted to reject a query with a `QueryTooComplex` exception. This provides a minimum level of interoperability.

8.2.5 Query framework (EPCIS query control interface)

The EPCIS Query Control Interface provides a general framework by which client applications may query EPCIS data. The interface provides both on-demand queries, in which an explicit request from a client causes a query to be executed and results returned in response, and standing queries, in which a client registers ongoing interest in a query and thereafter receives periodic delivery of results via the EPCIS Query Callback Interface without making further requests. These two modes are informally referred to as "pull" and "push," respectively.

The EPCIS Query Control Interface is defined below. An implementation of the Query Control Interface SHALL implement all of the methods defined below.

```
<<interface>>
EPCISQueryControlInterface
---
subscribe(queryName : String, params : QueryParams, dest : URI, controls
: SubscriptionControls, subscriptionID : String)
unsubscribe(subscriptionID : String)
poll(queryName : String, params : QueryParams) : QueryResults
getQueryNames() : List // of names
getSubscriptionIDs(queryName : String) : List // of Strings
getStandardVersion() : string
getVendorVersion() : string
<<extension point>>
```

Standing queries are made by making one or more subscriptions to a previously defined query using the `subscribe` method. Results will be delivered periodically via the Query Callback Interface to a specified destination, until the subscription is cancelled using the `unsubscribe` method. On-demand queries are made by executing a previously defined query using the `poll` method. Each invocation of the `poll` method returns a result directly to the caller. In either case, if the query is parameterised, specific settings for the parameters may be provided as arguments to `subscribe` or `poll`.

An implementation MAY provide one or more "pre-defined" queries. A pre-defined query is available for use by `subscribe` or `poll`, and is returned in the list of query names returned by `getQueryNames`, without the client having previously taken any action to define the query. In particular, EPCIS 1.0 does not support any mechanism by which a client can define a new query, and so pre-defined queries are the *only* queries available. See section 8.2.7 for specific pre-defined queries that SHALL be provided by an implementation of the EPCIS 1.0 Query Interface.

An implementation MAY permit a given query to be used with `poll` but not with `subscribe`. Generally, queries for event data may be used with both `poll` and `subscribe`, but queries for master data may be used only with `poll`. This is because `subscribe` establishes a periodic schedule for running a query multiple times, each time restricting attention to new events recorded since the last time the query was run. This mechanism cannot apply to queries for master data, because master data is presumed to be quasi-static and does not have anything corresponding to a record time.

The specification of these methods is as follows:

Method	Description
`subscribe`	Registers a subscriber for a previously defined query having the specified name. The `params` argument provides the values to be used for any named parameters defined by the query. The `dest` parameter specifies a destination where results from the query are to be delivered, via the Query Callback Interface. The `dest` parameter is a URI that both identifies a specific binding of the Query Callback Interface to use and specifies addressing information. The `controls` parameter controls how the subscription is to be processed; in particular, it specifies the conditions under which the query is to be invoked (e.g., specifying a periodic schedule). The `subscriptionID` is an arbitrary string that is copied into every response delivered to the specified destination, and otherwise not interpreted by the EPCIS service. The client may use the `subscriptionID` to identify from which subscription a given result was generated, especially when several subscriptions are made to the same destination. The `dest` argument may be null or empty, in which case the EPCIS implementation SHALL deliver results to a pre-arranged destination based on the authenticated identity of the caller; however, if the implementation does not have a destination pre-arranged for the caller, or does not permit this usage, it SHALL raise an `InvalidURIException` instead.
`unsubscribe`	Removes a previously registered subscription having the specified `subscriptionID`.
`poll`	Invokes a previously defined query having the specified name, returning the results. The `params` argument provides the values to be used for any named parameters defined by the query.
`getQueryNames`	Returns a list of all query names available for use with the `subscribe` and `poll` methods. This includes all pre-defined queries provided by the implementation, including those specified in section 8.2.7.
`getSubscriptionIDs`	Returns a list of all `subscriptionID`s currently subscribed to the specified named query.
`getStandardVersion`	Returns a string that identifies what version of the EPCIS specification this implementation complies with. The possible values for this string are defined by GS1. An implementation SHALL return a string corresponding to a version of this specification to which the implementation fully complies, and SHOULD return the string corresponding to the latest version with which it complies. To indicate compliance with this Version 2.0 of the EPCIS specification, the implementation SHALL return the string `2.0`.

Method	Description
`getVendorVersion`	Returns a string that identifies what vendor extensions this implementation provides. The possible values of this string and their meanings are vendor-defined, except that the empty string SHALL indicate that the implementation implements only standard functionality with no vendor extensions. When an implementation chooses to return a non-empty string, the value returned SHALL be a URI where the vendor is the owning authority. For example, this may be an HTTP URL whose authority portion is a domain name owned by the vendor, a URN having a URN namespace identifier issued to the vendor by IANA, an OID URN whose initial path is a Private Enterprise Number assigned to the vendor, etc.

This framework applies regardless of the content of a query. The detailed contents of a query, and the results as returned from `poll` or delivered to a subscriber via the Query Callback Interface, are defined in later sections of this document. This structure is designed to facilitate extensibility, as new types of queries may be specified and fit into this general framework.

An implementation MAY restrict the behaviour of any method according to authorisation decisions based on the authenticated client identity of the client making the request. For example, an implementation may limit the IDs returned by `getSubscriptionIDs` and recognised by `unsubscribe` to just those subscribers that were previously subscribed by the same client identity. This allows a single EPCIS service to be "partitioned" for use by groups of unrelated users whose data should be kept separate.

If a pre-defined query defines named parameters, values for those parameters may be supplied when the query is subsequently referred to using `poll` or `subscribe`. A `QueryParams` instance is simply a set of name/value pairs, where the names correspond to parameter names defined by the query, and the values are the specific values to be used for that invocation of (`poll`) or subscription to (`subscribe`) the query. If a `QueryParams` instance includes a name/value pair where the value is empty string, empty array or `null`, it SHALL be interpreted as though that query parameter were omitted altogether and disregarded.

The `poll` or `subscribe` method SHALL raise a `QueryParameterException` under any of the following circumstances:

- A parameter required by the specified query was omitted or was supplied with an empty value
- A parameter was supplied whose name does not correspond to any parameter name defined by the specified query
- Two parameters are supplied having the same name
- Any other constraint imposed by the specified query is violated. Such constraints may include restrictions on the range of values permitted for a given parameter, requirements that two or more parameters be mutually exclusive or must be supplied together, and so on. The specific constraints imposed by a given query are specified in the documentation for that query.

8.2.5.1 Subscription controls

Standing queries are subscribed to via the `subscribe` method. For each subscription, a `SubscriptionControls` instance defines how the query is to be processed.

```
SubscriptionControls
---
schedule : QuerySchedule // see Section 8.2.5.3
trigger : URI // specifies a trigger event known by the service
initialRecordTime : Time // see Section 8.2.5.2
reportIfEmpty : boolean
<<extension point>>
```

The fields of a `SubscriptionControls` instance are defined below.

Argument	Type	Description
`schedule`	`QuerySchedule`	(Optional) Defines the periodic schedule on which the query is to be executed. See section 8.2.5.3. Exactly one of `schedule` or `trigger` is required; if both are specified or both are omitted, the implementation SHALL raise a `SubscriptionControlsException`.
`trigger`	`URI`	(Optional) Specifies a triggering event known to the EPCIS service that will serve to trigger execution of this query. The `available` trigger URIs are service-dependent. Exactly one of `schedule` or `trigger` is required; if both are specified or both are omitted, the implementation SHALL raise a `SubscriptionControlsException`.
`initialRecordTime`	`Time`	(Optional) Specifies a time used to constrain what events are considered when processing the query when it is executed for the first time. See section 8.2.5.2. If omitted, defaults to the time at which the subscription is created.
`reportIfEmpty`	`boolean`	If true, a `QueryResults` instance is always sent to the subscriber when the query is executed. If false, a `QueryResults` instance is sent to the subscriber only when the results are non-empty.

8.2.5.2 Automatic limitation based on event record time

Each subscription to a query results in the query being executed many times in succession, the timing of each execution being controlled by the specified `schedule` or being triggered by a triggering condition specified by `trigger`. Having multiple executions of the same query is only sensible if each execution is limited in scope to new event data generated since the last execution – otherwise, the same events would be returned more than once. However, the time constraints cannot be specified explicitly in the query or query parameters, because these do not change from one execution to the next.

For this reason, an EPCIS service SHALL constrain the scope of each query execution for a subscribed query in the following manner. The first time the query is executed for a given subscription, the only events considered are those whose `recordTime` field is greater than or equal to `initialRecordTime` specified when the subscription was created. For each execution of the query following the first, the only events considered are those whose `recordTime` field is greater than or equal to the time when the query was last executed. It is implementation dependent as to the extent that failure to deliver query results to the subscriber affects this calculation; implementations SHOULD make best efforts to insure reliable delivery of query results so that a subscriber does not miss any data. The query or query parameters may specify additional constraints upon record time; these are applied after restricting the universe of events as described above.

Non-Normative: Explanation: one possible implementation of this requirement is that the EPCIS service maintains a `minRecordTime` value for each subscription that exists. The `minRecordTime` for a given subscription is initially set to `initialRecordTime`, and updated to the current time each time the query is executed for that subscription. Each time the query is executed, the only events considered are those whose `recordTime` is greater than or equal to `minRecordTime` for that subscription.

8.2.5.3 Query schedule

A `QuerySchedule` may be specified to specify a periodic schedule for query execution for a specific subscription. Each field of `QuerySchedule` is a string that specifies a pattern for matching some part of the current time. The query will be executed each time the current date and time matches the specification in the `QuerySchedule`.

Each `QuerySchedule` field is a string, whose value must conform to the following grammar:

```
QueryScheduleField ::= Element ( ",” Element )*

Element ::= Number | Range

Range ::= "[" Number "-" Number "]”

Number ::= Digit+

Digit ::= "0” | "1” | "2” | "3” | "4”
        | "5” | "6” | "7” | "8” | "9”
```

Each `Number` that is part of the query schedule field value must fall within the legal range for that field as specified in the table below. An EPCIS implementation SHALL raise a `SubscriptionControlsException` if any query schedule field value does not conform to the grammar above, or contains a `Number` that falls outside the legal range, or includes a `Range` where the first `Number` is greater than the second `Number`.

The `QuerySchedule` specifies a periodic sequence of time values (the “query times”). A query time is any time value that matches the `QuerySchedule`, according to the following rule:

- Given a time value, extract the second, minute, hour (0 through 23, inclusive), dayOfMonth (1 through 31, inclusive), and dayOfWeek (1 through 7, inclusive, denoting Monday through Sunday). This calculation is to be performed relative to a time zone chosen by the EPCIS Service.
- The time value matches the `QuerySchedule` if each of the values extracted above matches (as defined below) the corresponding field of the `QuerySchedule`, for all `QuerySchedule` fields that are not omitted.
- A value extracted from the time value matches a field of the `QuerySchedule` if it matches any of the comma-separated `Element`s of the query schedule field.
- A value extracted from the time value matches an `Element` of a query schedule field if
- the `Element` is a `Number` and the value extracted from the time value is equal to the `Number`; or
- the `Element` is a `Range` and the value extracted from the time value is greater than or equal to the first `Number` in the `Range` and less than or equal to the second `Number` in the `Range`.

See examples following the table below.

An EPCIS implementation SHALL interpret the `QuerySchedule` as a client’s statement of when it would like the query to be executed, and SHOULD make reasonable efforts to adhere to that schedule. An EPCIS implementation MAY, however, deviate from the requested schedule according to its own policies regarding server load, authorisation, or any other reason. If an EPCIS implementation knows, at the time the `subscribe` method is called, that it will not be able to honour the specified `QuerySchedule` without deviating widely from the request, the EPCIS implementation SHOULD raise a `SubscriptionControlsException` instead.

Non-Normative: Explanation: The `QuerySchedule`, taken literally, specifies the exact timing of query execution down to the second. In practice, an implementation may not wish to or may not be able to honour that request precisely, but can honour the general intent. For example, a `QuerySchedule`

may specify that a query be executed every hour on the hour, while an implementation may choose to execute the query every hour plus or minus five minutes from the top of the hour. The paragraph above is intended to give implementations latitude for this kind of deviation.

In any case, the automatic handling of `recordTime` as specified earlier SHALL be based on the actual time the query is executed, whether or not that exactly matches the `QuerySchedule`.

The field of a `QuerySchedule` instance are as follows.

Argument	Type	Description
`second`	`String`	(Optional) Specifies that the query time must have a matching seconds value. The range for this parameter is 0 through 59, inclusive.
`minute`	`String`	(Optional) Specifies that the query time must have a matching minute value. The range for this parameter is 0 through 59, inclusive.
`hour`	`String`	(Optional) Specifies that the query time must have a matching hour value. The range for this parameter is 0 through 23, inclusive, with 0 denoting the hour that begins at midnight, and 23 denoting the hour that ends at midnight.
`dayOfMonth`	`String`	(Optional) Specifies that the query time must have a matching day of month value. The range for this parameter is 1 through 31, inclusive. (Values of 29, 30, and 31 will only match during months that have at least that many days.)
`month`	`String`	(Optional) Specifies that the query time must have a matching month value. The range for this parameter is 1 through 12, inclusive.
`dayOfWeek`	`String`	(Optional) Specifies that the query time must have a matching day of week value. The range for this parameter is 1 through 7, inclusive, with 1 denoting Monday, 2 denoting Tuesday, and so forth, up to 7 denoting Sunday. Explanation (non-normative): this numbering scheme is consistent with ISO 8601 [ISO8601].

8.2.5.3.1 Query schedule examples (Non-Normative)

Here are some examples of `QuerySchedule` and what they mean.

Example 1

QuerySchedule
  second = "0"
  minute = "0"
  all other fields omitted

This means "run the query once per hour, at the top of the hour." If the reportIfEmpty argument to subscribe is false, then this does not necessarily cause a report to be sent each hour – a report would be sent within an hour of any new event data becoming available that matches the query.

Example 2

QuerySchedule
  second = "0"
  minute = "30"
  hour = "2"
  all other fields omitted

This means "run the query once per day, at 2:30 am."

Example 3

QuerySchedule
second = "0"
minute = "0"
dayOfWeek = "[1-5]"

This means "run the query once per hour at the top of the hour, but only on weekdays."

Example 4

QuerySchedule
hour = "2"
all other fields omitted

This means "run the query once per second between 2:00:00 and 2:59:59 each day." This example illustrates that it usually not desirable to omit a field of finer granularity than the fields that are specified.

8.2.5.4 QueryResults

A `QueryResults` instance is returned synchronously from the `poll` method of the EPCIS Query Control Interface, and also delivered asynchronously to a subscriber of a standing query via the EPCIS Query Callback Interface.

```
QueryResults
---
queryName : string
subscriptionID : string
resultsBody : QueryResultsBody
<<extension point>>
```

The fields of a `QueryResults` instance are defined below.

Field	Type	Description
`queryName`	`String`	This field SHALL contain the name of the query (the `queryName` argument that was specified in the call to `poll` or `subscribe`).
`subscriptionID`	`string`	(Conditional) When a `QueryResults` instance is delivered to a subscriber as the result of a standing query, `subscriptionID` SHALL contain the same string provided as the `subscriptionID` argument the call to `subscribe`. When a `QueryResults` instance is returned as the result of a `poll` method, this field SHALL be omitted.
`resultsBody`	`QueryResultsBody`	The information returned as the result of a query. The exact type of this field depends on which query is executed. Each of the predefined queries in section 8.2.7 specifies the corresponding type for this field.

8.2.6 Error conditions

Methods of the EPCIS Query Control API signal error conditions to the client by means of exceptions. The following exceptions are defined. All the exception types in the following table are extensions of a common `EPCISException` base type, which contains one required string element giving the reason for the exception.

Exception Name	Meaning
SecurityException	The operation was not permitted due to an access control violation or other security concern. This includes the case where the service wishes to deny authorisation to execute a particular operation based on the authenticated client identity. The specific circumstances that may cause this exception are implementation-specific, and outside the scope of this specification.
DuplicateNameException	The specified query name already exists.
QueryValidationException	The specified query is invalid; e.g., it contains a syntax error.
QueryParameterException	One or more query parameters are invalid, including any of the following situations: ▪ the parameter name is not a recognised parameter for the specified query ▪ the value of a parameter is of the wrong type or out of range ▪ two or more query parameters have the same parameter name
QueryTooLargeException	An attempt to execute a query resulted in more data than the service was willing to provide.
QueryTooComplexException	The specified query parameters, while otherwise valid, implied a query that was more complex than the service was willing to execute.
InvalidURIException	The URI specified for a subscriber cannot be parsed, does not name a scheme recognised by the implementation, or violates rules imposed by a particular scheme.
SubscriptionControlsException	The specified subscription controls was invalid; e.g., the schedule parameters were out of range, the trigger URI could not be parsed or did not name a recognised trigger, etc.
NoSuchNameException	The specified query name does not exist.
NoSuchSubscriptionException	The specified subscriptionID does not exist.
NoSuchResourceException	The specified resource does not exist.
DuplicateSubscriptionException	The specified subscriptionID is identical to a previous subscription that was created and not yet unsubscribed.
SubscribeNotPermittedException	The specified query name may not be used with subscribe, only with poll.
ValidationException	The input to the operation was not syntactically valid according to the syntax defined by the binding. Each binding specifies the particular circumstances under which this exception is raised.
ImplementationException	A generic exception thrown by the implementation for reasons that are implementation-specific. This exception contains one additional element: a severity member whose values are either ERROR or SEVERE. ERROR indicates that the EPCIS implementation is left in the same state it had before the operation was attempted. SEVERE indicates that the EPCIS implementation is left in an indeterminate state.

Note that the REST interface does not implement the following exceptions:

The exceptions that may be thrown by each method of the EPCIS Query Control Interface are indicated in the table below:

EPCIS Method	Exceptions

`getQueryNames`	`SecurityException`
	`ValidationException`
	`ImplementationException`

`subscribe`	`NoSuchNameException`
	`InvalidURIException`
	`DuplicateSubscriptionException`
	`QueryParameterException`
	`QueryTooComplexException`
	`SubscriptionControlsException`
	`SubscribeNotPermittedException`
	`SecurityException`
	`ValidationException`
	`ImplementationException`

`unsubscribe`	`NoSuchSubscriptionException`
	`SecurityException`
	`ValidationException`
	`ImplementationException`

`poll`	`NoSuchNameException`
	`QueryParameterException`
	`QueryTooComplexException`
	`QueryTooLargeException`
	`SecurityException`
	`ValidationException`
	`ImplementationException`

`getSubscriptionIDs`	`NoSuchNameException SecurityException`
	`ValidationException`
	`ImplementationException`

`getStandardVersion`	`SecurityException`
	`ValidationException`
	`ImplementationException`

`getVendorVersion`	`SecurityException`
	`ValidationException`
	`ImplementationException`

In addition to exceptions thrown from methods of the EPCIS Query Control Interface as enumerated above, an attempt to execute a standing query may result in a `QueryTooLargeException` or an `ImplementationException` being sent to a subscriber via the EPCIS Query Callback Interface instead of a normal query result. In this case, the `QueryTooLargeException` or `ImplementationException` SHALL include, in addition to the reason string, the query name and the `subscriptionID` as specified in the `subscribe` call that created the standing query.

8.2.7 Predefined queries for EPCIS

In EPCIS, no query language is provided by which a client may express an arbitrary query for data. Instead, an EPCIS implementation SHALL provide the following predefined queries, which a client may invoke using the `poll` and `subscribe` methods of the EPCIS Query Control Interface. Each poll or subscribe call may include parameters via the params argument. The predefined queries defined in this section each have a large number of optional parameters; by appropriate choice of parameters a client can achieve a variety of effects.

The parameters for each predefined query and what results it returns are specified in this section. An implementation of EPCIS is free to use any internal representation for

data it wishes, and implement these predefined queries using any database or query technology it chooses, so long as the results seen by a client are consistent with this specification.

8.2.7.1 SimpleEventQuery

This query is invoked by specifying the string `SimpleEventQuery` as the `queryName` argument to `poll` or `subscribe`. The result is a `QueryResults` instance whose body contains a (possibly empty) list of `EPCISEvent` instances. Unless constrained by the `eventType` parameter, each property of the result list could be of any event type; i.e., `ObjectEvent`, `AggregationEvent`, `TransactionEvent`, or any extension event type that is a subclass of `EPCISEvent`.

The `SimpleEventQuery` SHALL be available via both `poll` and `subscribe`; that is, an implementation SHALL NOT raise `SubscribeNotPermittedException` when `SimpleEventQuery` is specified as the `queryName` argument to `subscribe`.

The `SimpleEventQuery` is defined to return a set of events that matches the criteria specified in the query parameters (as specified below). When returning events that were captured via the EPCIS Capture Interface, each event that is selected to be returned SHALL be identical to the originally captured event, subject to the provisions of authorisation (section 8.2.2), the inclusion of the `recordTime` field, and any necessary conversions to and from an abstract internal representation. For any event field defined to hold an unordered list, however, an EPCIS implementation MAY preserve the order.

The parameters for this query are as follows. None of these parameters is required (though in most cases, a query will include at least one query parameter).

Parameter name	Parameter value type	Meaning
`eventType`	List of String	If specified, the result will only include events whose type matches one of the types specified in the parameter value. Each property of the parameter value may be one of the following strings: `ObjectEvent`, `AggregationEvent`, `TransactionEvent`, `TransformationEvent` or `AssociationEvent`. A property of the parameter value may also be the name of an extension event type. If omitted, all event types will be considered for inclusion in the result.
`GE_eventTime`	DateTimeStamp	If specified, only events with `eventTime` greater than or equal to the specified value will be included in the result. If omitted, events are included regardless of their `eventTime` (unless constrained by the `LT_eventTime` parameter).
`LT_eventTime`	DateTimeStamp	If specified, only events with `eventTime` less than the specified value will be included in the result. If omitted, events are included regardless of their `eventTime` (unless constrained by the `GE_eventTime` parameter).

Parameter name	Parameter value type	Meaning
GE_recordTime	DateTimeStamp	If provided, only events with recordTime greater than or equal to the specified value will be returned. The automatic limitation based on event record time (section 8.2.5.2) may implicitly provide a constraint similar to this parameter. If omitted, events are included regardless of their recordTime, other than automatic limitation based on event record time (section 8.2.5.2).
LT_recordTime	DateTimeStamp	If provided, only events with recordTime less than the specified value will be returned. If omitted, events are included regardless of their recordTime (unless constrained by the GE_recordTime parameter or the automatic limitation based on event record time).
EQ_action	List of String	If specified, the result will only include events that (a) have an action field; and where (b) the value of the action field matches one of the specified values. The properties of the value of this parameter each must be one of the strings ADD, OBSERVE, or DELETE; if not, the implementation SHALL raise a QueryParameterException. If omitted, events are included regardless of their action field.
EQ_bizStep	List of URIs	If specified, the result will only include events that (a) have a non-null bizStep field; and where (b) the value of the bizStep field matches one of the specified values. If this parameter is omitted, events are returned regardless of the value of the bizStep field or whether the bizStep field exists at all.
EQ_disposition	List of URIs	Like the EQ_bizStep parameter, but for the disposition field.
EQ_persistentDisposition_set	List of URIs	Like the EQ_bizStep parameter, but for the persistentDisposition set field.
EQ_persistentDisposition_unset	List of URIs	Like the EQ_bizStep parameter, but for the persistentDisposition unset field.

Parameter name	Parameter value type	Meaning
`EQ_readPoint`	List of URIs	If specified, the result will only include events that (a) have a non-null `readPoint` field; and where (b) the value of the `readPoint` field matches one of the specified URIs. If this parameter and `WD_readPoint` are both omitted, events are returned regardless of the value of the `readPoint` field or whether the `readPoint` field exists at all.
`EQ_readPoint_`*`fieldname`*	List of String	Analogous to `EQ_fieldname`, but matches events whose `readPoint` contains a field having the specified `fieldname` whose value matches one of the specified values.
`EQ_readPoint_`*`fieldname`*	Int Float DateTimeStamp Double	Like `EQ_readPoint_`*`fieldname`* as described above, but may be applied to a field of type Int, Float, Double or Time. The result will include events whose `readPoint` (a) has a field named `fieldname`; and where (b) the type of the field matches the type of this parameter (Int, Float, Double or Time); and where (c) the value of the field is equal to the specified value. `fieldname` is constructed as for `EQ_readPoint_fieldname`.
`EQ_INNER_readPoint_`*`fieldname`*	List of String	Analogous to `EQ_INNER fieldname`, but matches inner extension fields; that is, any field nested within a top-level extension element. Note that a matching inner field may exist within more than one top-level field or may occur more than once within a single top-level field; this parameter matches if at least one matching occurrence is found anywhere in the event (except at top-level).

Parameter name	Parameter value type	Meaning
`EQ_INNER_readPoint_`*`fieldname`*	Int Float DateTimeStamp Double	Like `EQ_INNER_readPoint_`*`fieldname`* as described above, but may be applied to a field of type Int, Float, Double or Time. The result will include events whose `readPoint` (a) has an inner extension (nested) field named `fieldname`; and where (b) the type of the field matches the type of this parameter (Int, Float, Double or Time); and where (c) the value of the field is equal to the specified value. `fieldname` is constructed as for `EQ_INNER_readPoint_fieldname`.
`WD_readPoint`	List of URIs	If specified, the result will only include events that (a) have a non-null `readPoint` field; and where (b) the value of the `readPoint` field matches one of the specified URIs, or is a direct or indirect descendant of one of the specified values. The meaning of "direct or indirect descendant" is specified by master data, as described in section 6.5. (`WD` is an abbreviation for "with descendants.") If this parameter and `EQ_readPoint` are both omitted, events are returned regardless of the value of the `readPoint` field or whether the `readPoint` field exists at all.
`EQ_bizLocation`	List of URIs	Like the `EQ_readPoint` parameter, but for the `bizLocation` field.
`EQ_bizLocation_`*`fieldname`*	List of String	Analogous to `EQ_readPoint_fieldname`, but matches events whose `bizLocation` contains a field having the specified `fieldname` whose value matches one of the specified values.

Parameter name	Parameter value type	Meaning
`EQ_bizLocation_`*`fieldname`*	Int Float DateTimeStamp Double	Like `EQ_bizLocation_`*`fieldname`* as described above, but may be applied to a field of type Int, Float, Double or Time. The result will include events whose `bizLocation` (a) has a field named `fieldname`; and where (b) the type of the field matches the type of this parameter (Int, Float, Double or Time); and where (c) the value of the field is equal to the specified value. `fieldname` is constructed as for `EQ_bizLocation_fieldname`.
`EQ_INNER_bizLocation_`*`fieldname`*	List of String	Analogous to `EQ_INNER_bizLocation_fieldname`, but matches inner extension fields; that is, any field nested within a top-level extension element. Note that a matching inner field may exist within more than one top-level field or may occur more than once within a single top-level field; this parameter matches if at least one matching occurrence is found anywhere in the event (except at top-level).
`EQ_INNER_bizLocation_`*`fieldname`*	Int Float DateTimeStamp Double	Like `EQ_INNER_bizLocation_`*`fieldname`* as described above, but may be applied to a field of type Int, Float, Double or Time. The result will include events whose `bizLocation` (a) has an inner extension (nested) field named `fieldname`; and where (b) the type of the field matches the type of this parameter (Int, Float, Double or Time); and where (c) the value of the field is equal to the specified value. `fieldname` is constructed as for `EQ_INNER_bizLocation_fieldname`.
`WD_bizLocation`	List of URIs	Like the `WD_readPoint` parameter, but for the `bizLocation` field.

Parameter name	Parameter value type	Meaning
`EQ_bizTransaction_`*`type`*	List of URIs	This is not a single parameter, but a family of parameters. If a parameter of this form is specified, the result will only include events that (a) include a `bizTransactionList`; (b) where the business transaction list includes an entry whose `type` subfield is equal to `type` extracted from the name of this parameter; and (c) where the `bizTransaction` subfield of that entry is equal to one of the values specified in this parameter.
`EQ_source_`*`type`*	List of URIs	This is not a single parameter, but a family of parameters. If a parameter of this form is specified, the result will only include events that (a) include a `sourceList`; (b) where the source list includes an entry whose `type` subfield is equal to `type` extracted from the name of this parameter; and (c) where the `source` subfield of that entry is equal to one of the values specified in this parameter.
`EQ_destination_`*`type`*	List of URIs	This is not a single parameter, but a family of parameters. If a parameter of this form is specified, the result will only include events that (a) include a `destinationList`; (b) where the destination list includes an entry whose `type` subfield is equal to `type` extracted from the name of this parameter; and (c) where the `destination` subfield of that entry is equal to one of the values specified in this parameter.
`EQ_transformationID`	List of URIs	If this parameter is specified, the result will only include events that (a) have a `transformationID` field (that is, `TransformationEvent`s or extension event type that extend `TransformationEvent`); and where (b) the `transformationID` field is equal to one of the values specified in this parameter.

Parameter name	Parameter value type	Meaning
MATCH_epc	List of URIs	If this parameter is specified, the result will only include events that (a) have an epcList or a childEPCs field (that is, ObjectEvent, AggregationEvent, TransactionEvent, AssociationEvent or extension event types that extend one of those event types); and where (b) one of the EPCs listed in the epcList or childEPCs field (depending on event type) matches one of the URIs specified in this parameter, where the meaning of "matches" is as specified in section 8.2.7.1.1. If this parameter is omitted, events are included regardless of their epcList or childEPCs field or whether the epcList or childEPCs field exists.
MATCH_parentID	List of URIs	Like MATCH_epc, but matches the parentID field of AggregationEvent, the parentID field of TransactionEvent, the parentID field of AssociationEvent and extension event types that extend those event types. The meaning of "matches" is as specified in section 8.2.7.1.1.
MATCH_inputEPC	List of URIs	If this parameter is specified, the result will only include events that (a) have an inputEPCList (that is, TransformationEvent or an extension event type that extends TransformationEvent); and where (b) one of the EPCs listed in the inputEPCList field matches one of the URIs specified in this parameter. The meaning of "matches" is as specified in section 8.2.7.1.1. If this parameter is omitted, events are included regardless of their inputEPCList field or whether the inputEPCList field exists.

Parameter name	Parameter value type	Meaning
MATCH_outputEPC	List of URIs	If this parameter is specified, the result will only include events that (a) have an outputEPCList (that is, TransformationEvent or an extension event type that extends TransformationEvent); and where (b) one of the EPCs listed in the outputEPCList field matches one of the URIs specified in this parameter. The meaning of "matches" is as specified in section 8.2.7.1.1. If this parameter is omitted, events are included regardless of their outputEPCList field or whether the outputEPCList field exists.
MATCH_anyEPC	List of URIs	If this parameter is specified, the result will only include events that (a) have an epcList field, a childEPCs field, a parentID field, an inputEPCList field, or an outputEPCList field (that is, ObjectEvent, AggregationEvent, TransactionEvent, TransformationEvent, AssociationEvent or extension event types that extend one of those event types); and where (b) the parentID field or one of the EPCs listed in the epcList, childEPCs, inputEPCList, or outputEPCList field (depending on event type) matches one of URIs specified in this parameter. The meaning of "matches" is as specified in section 8.2.7.1.1.
MATCH_epcClass	List of URIs	If this parameter is specified, the result will only include events that (a) have a quantityList or a childQuantityList field (that is, ObjectEvent, AggregationEvent, TransactionEvent, AssociationEvent or extension event types that extend one of those event types); and where (b) one of the EPC classes listed in the quantityList or childQuantityList field (depending on event type) matches one of the EPC patterns or URIs specified in this parameter. The result will also include QuantityEvents whose epcClass field matches one of the URIs specified in this parameter. The meaning of "matches" is as specified in section 8.2.7.1.1.

Parameter name	Parameter value type	Meaning
MATCH_inputEPCClass	List of URIs	If this parameter is specified, the result will only include events that (a) have an inputQuantityList field (that is, TransformationEvent or extension event types that extend it); and where (b) one of the EPC classes listed in the inputQuantityList field (depending on event type) matches one of the EPC patterns or URIs specified in this parameter. The meaning of "matches" is as specified in section 8.2.7.1.1.
MATCH_outputEPCClass	List of URIs	If this parameter is specified, the result will only include events that (a) have an outputQuantityList field (that is, TransformationEvent or extension event types that extend it); and where (b) one of the EPC classes listed in the outputQuantityList field (depending on event type) matches one of the URIs specified in this parameter. The meaning of "matches" is as specified in section 8.2.7.1.1.
MATCH_anyEPCClass	List of URIs	If this parameter is specified, the result will only include events that (a) have a quantityList, childQuantityList, inputQuantityList, or outputQuantityList field (that is, ObjectEvent, AggregationEvent, TransactionEvent, TransformationEvent, AssociationEvent or extension event types that extend one of those event types); and where (b) one of the EPC classes listed in any of those fields matches one of the EPC patterns or URIs specified in this parameter. The result will also include QuantityEvents whose epcClass field matches one of the URIs specified in this parameter. The meaning of "matches" is as specified in section 8.2.7.1.1.
EQ_quantity	Double	(DEPCRECATED in EPCIS 1.1, REPURPOSED in EPCIS 2.0) If this parameter is specified, the result will only include events that (a) have a quantity field as part of a QuantityElement; and where (b) the quantity field is equal to the specified parameter.

Parameter name	Parameter value type	Meaning
EQ_quantity_*uom*	Double	If this parameter is specified, the result will only include events that (a) have a quantity and a uom field as part of a QuantityElement; and where (b) a pair of quantity and uom is equal to or – in case the query includes a uom value that is different from those in the events – corresponds to the specified parameter. If omitted, events are in included regardless of the values of their quantity and uom fields.
GT_quantity	Double	(DEPCRECATED in EPCIS 1.1, REPURPOSED in EPCIS 2.0) Like EQ_quantity, but includes events whose quantity field is greater than the specified parameter.
GT_quantity_*uom*	Double	Like EQ_quantity_uom, but includes events whose quantity-*uom*-pair (or a corresponding quantity-*uom*-pair when an alternative uom is applied) is greater than the specified parameter.
GE_quantity	Double	(DEPCRECATED in EPCIS 1.1, REPURPOSED in EPCIS 2.0) Like EQ_quantity, but includes events whose quantity field is greater than or equal to the specified parameter.
GE_quantity_*uom*	Double	Like EQ_quantity_uom, but includes events whose quantity-*uom*-pair (or a corresponding quantity-*uom*-pair when an alternative uom is applied) is greater than or equal to the specified parameter.
LT_quantity	Double	(DEPCRECATED in EPCIS 1.1, REPURPOSED in EPCIS 2.0) Like EQ_quantity, but includes events whose quantity field is less than the specified parameter.
LT_quantity_*uom*	Double	Like EQ_quantity_uom, but includes events whose quantity-*uom*-pair (or a corresponding quantity-*uom*-pair when an alternative uom is applied) is less than the specified parameter.
LE_quantity	Double	(DEPCRECATED in EPCIS 1.1, REPURPOSED in EPCIS 2.0) Like EQ_quantity, but includes events whose quantity field is less than or equal to the specified parameter.

Parameter name	Parameter value type	Meaning
LE_quantity_*uom*	Double	Like EQ_quantity_uom, but includes events whose quantity-*uom*-pair (or a corresponding quantity-*uom*-pair when an alternative uom is applied) is less than or equal to the specified parameter.
EQ_*fieldname*	List of String	This is not a single parameter, but a family of parameters. If a parameter of this form is specified, the result will only include events that (a) have a top-level extension field named *fieldname* whose type is either String or a vocabulary type; and where (b) the value of that field matches one of the values specified in this parameter. fieldname is the fully qualified name of a top-level extension field. The name of an extension field is an XML qname; that is, a pair consisting of an XML namespace URI and a name. The name of the corresponding query parameter is constructed by concatenating the following: the string EQ_, the namespace URI for the extension field, a pound sign (#), and the name of the extension field. "Top level" means that the matching extension data field must be nested as an immediate child attribute of the containing EPCIS event, not a data field nested within a top-level event extension or class. See EQ_INNER_fieldname for querying data fields nested within extension elements / classes.
EQ_*fieldname*	Int Float DateTimeStamp Double	Like EQ_*fieldname* as described above, but may be applied to a field of type Int, Float, Double or Time. The result will include events that (a) have a field named fieldname; and where (b) the type of the field matches the type of this parameter (Int, Float, Double or Time); and where (c) the value of the field is equal to the specified value. fieldname is constructed as for EQ_fieldname.

Parameter name	Parameter value type	Meaning
`GT_`*`fieldname`*	Int Float DateTimeStamp Double	Like `EQ_fieldname` as described above, but may be applied to a field of type Int, Float, Double or Time. The result will include events that (a) have a field named `fieldname`; and where (b) the type of the field matches the type of this parameter (Int, Float, Double or Time); and where (c) the value of the field is greater than the specified value. `fieldname` is constructed as for `EQ_fieldname`.
`GE_`*`fieldname`* `LT_`*`fieldname`* `LE_`*`fieldname`*	Int Float DateTimeStamp Double	Analogous to `GT_fieldname`
`EQ_ILMD_`*`fieldname`*	List of String	This is not a single parameter, but a family of parameters. Analogous to `EQ_fieldname`, but matches events whose ILMD area (section 7.3.8) contains a top-level field having the specified `fieldname` whose value matches one of the specified values. "Top level" means that the matching ILMD field must be an immediate child of the `<ilmd>` element, not an element nested within such an element. See `EQ_INNER_ILMD_`*`fieldname`* for querying inner extension elements.
`EQ_ILMD_`*`fieldname`*	Int Float DateTimeStamp Double	Like `EQ_ILMD_`*`fieldname`* as described above, but may be applied to a field of type Int, Float, Double or Time. The result will include events that (a) have an ILMD field named `fieldname`; and where (b) the type of the field matches the type of this parameter (Int, Float, Double or Time); and where (c) the value of the field is equal to the specified value. `fieldname` is constructed as for `EQ_ILMD_fieldname`.

Parameter name	Parameter value type	Meaning
GT_ILMD_*fieldname* GE_ILMD_*fieldname* LT_ILMD_*fieldname* LE_ILMD_*fieldname*	Int Float DateTimeStamp Double	Analogous to EQ_fieldname, GT_fieldname, GE_fieldname, GE_fieldname, LT_fieldname, and LE_fieldname, respectively, but matches events whose ILMD (section 7.3.8) contains a field having the specified fieldname whose integer, float, double-precision float or time value matches the specified value according to the specified relational operator.
EQ_INNER_*fieldname*	List of String	Analogous to EQ_fieldname, but matches inner extension fields; that is, any XML field nested at any level within a top-level extension element. Note that a matching inner field may exist within more than one top-level element or may occur more than once within a single top-level element; this parameter matches if at least one matching occurrence is found anywhere in the event (except at top-level). Note that unlike a top-level extension element, an inner extension element may have a null XML namespace. To match such an inner element, the empty string is used in place of the XML namespace when constructing the query parameter name. For example, to match inner element <elt1> with no XML namespace, the query parameter would be EQ_INNER_#elt1.
EQ_INNER_*fieldname*	Int Float DateTimeStamp Double	Like EQ_INNER_*fieldname* as described above, but may be applied to a field of type Int, Float, Double or Time. The result will include events that (a) have an inner extension (nested) field named fieldname; and where (b) the type of the field matches the type of this parameter (Int, Float, Double or Time); and where (c) the value of the field is equal to the specified value. fieldname is constructed as for EQ_INNER_fieldname.
GT_INNER_*fieldname* GE_INNER_*fieldname* LT_INNER_*fieldname* LE_INNER_*fieldname*	Int Float DateTimeStamp Double	Like EQ_INNER_fieldname as described above, but may be applied to a field of type Int, Float, Double or Time.

Parameter name	Parameter value type	Meaning
EQ_INNER_ILMD_*fieldname*	List of String	Analogous to EQ_ILMD_fieldname, but matches inner ILMD elements; that is, any XML field nested at any level within a top-level ILMD element. Note that a matching inner field may exist within more than one top-level element or may occur more than once within a single top-level element; this parameter matches if at least one matching occurrence is found anywhere in the ILMD section (except at top-level).
GT_INNER_ILMD_*fieldname* GE_INNER_ILMD_*fieldname* LT_INNER_ILMD_*fieldname* LE_INNER_ILMD_*fieldname*	Int Float DateTimeStamp Double	Like EQ_INNER_ILMD_fieldname as described above, but may be applied to a field of type Int, Float, Double or Time.
EXISTS_*fieldname*	Void	Like EQ_fieldname as described above, but may be applied to a field of any type (including complex types). The result will include events that have a non-empty field named fieldname. Fieldname is constructed as for EQ_fieldname. Note that the value for this query parameter is ignored.
EXISTS_INNER_*fieldname*	Void	Like EXISTS_fieldname as described above, but includes events that have a non-empty inner extension field named *fieldname*. Note that the value for this query parameter is ignored.
EXISTS_ILMD_*fieldname*	Void	Like EXISTS_fieldname as described above, but events that have a non-empty field named fieldname in the ILMD area (section 7.3.8). Fieldname is constructed as for EQ_ILMD_fieldname. Note that the value for this query parameter is ignored.
EXISTS_INNER_ILMD_*fieldname*	Void	Like EXISTS_ILMD_fieldname as described above, but includes events that have a non-empty inner extension field named *fieldname* within the ILMD area. Note that the value for this query parameter is ignored.

Parameter name	Parameter value type	Meaning
`HASATTR_`*`fieldname`*	List of String	This is not a single parameter, but a family of parameters. If a parameter of this form is specified, the result will only include events that (a) have a field named `fieldname` whose type is a vocabulary type; and (b) where the value of that field is a vocabulary element for which master data is available; and (c) the master data has a non-null attribute whose name matches one of the values specified in this parameter. `Fieldname` is the fully qualified name of a field. For a standard field, this is simply the field name; e.g., `bizLocation`. For an extension field, the name of an extension field is an XML qname; that is, a pair consisting of an XML namespace URI and a name. The name of the corresponding query parameter is constructed by concatenating the following: the string `HASATTR_`, the namespace URI for the extension field, a pound sign (#), and the name of the extension field.

Parameter name	Parameter value type	Meaning
`EQ_ATTR_`*`fieldname_attrname`	List of String	This is not a single parameter, but a family of parameters. If a parameter of this form is specified, the result will only include events that (a) have a field named `fieldname` whose type is a vocabulary type; and (b) where the value of that field is a vocabulary element for which master data is available; and (c) the master data has a non-null attribute named `attrname`; and (d) where the value of that attribute matches one of the values specified in this parameter. `fieldname` is constructed as for `HASATTR_fieldname`. The implementation MAY raise a `QueryParameterException` if `fieldname` or `attrname` includes an underscore character. Explanation (non-normative): because the presence of an underscore in fieldname or attrname presents an ambiguity as to where the division between fieldname and attrname lies, an implementation is free to reject the query parameter if it cannot disambiguate.
`EQ_ATTR_`*`fieldname_attrname`	Int Float DateTimeStamp Double	Like `EQ_ATTR_`*`fieldname_attrname` as described above, but may be applied to a field of type Int, Float, Double or Time. The result will only include events that (a) have a field named `fieldname` whose type is a vocabulary type; and (b) where the value of that field is a vocabulary element for which master data is available; and (c) the master data has a non-null attribute named `attrname`; and where (b) the type of the field matches the type of this parameter (Int, Float, Double or Time); and where (c) the value of the field is equal to the specified value. `fieldname_attrname` is constructed as for `EQ_ATTR_fieldname_attrname`.

Parameter name	Parameter value type	Meaning
EQ_eventID	List of URIs	If this parameter is specified, the result will only include events that (a) have a non-null eventID field; and where (b) the eventID field is equal to one of the values specified in this parameter. If this parameter is omitted, events are returned regardless of the value of the eventID field or whether the eventID field exists at all.
EXISTS_errorDeclaration	Void	If this parameter is specified, the result will only include events that contain an ErrorDeclaration. If this parameter is omitted, events are returned regardless of whether they contain an ErrorDeclaration.
GE_errorDeclarationTime	DateTimeStamp	If this parameter is specified, the result will only include events that (a) contain an ErrorDeclaration; and where (b) the value of the errorDeclarationTime field is greater than or equal to the specified value. If this parameter is omitted, events are returned regardless of whether they contain an ErrorDeclaration or what the value of the errorDeclarationTime field is.
LT_errorDeclarationTime	DateTimeStamp	If this parameter is specified, the result will only include events that (a) contain an ErrorDeclaration; and where (b) the value of the errorDeclarationTime field is less than to the specified value. If this parameter is omitted, events are returned regardless of whether they contain an ErrorDeclaration or what the value of the errorDeclarationTime field is.
EQ_errorReason	List of URIs	If this parameter is specified, the result will only include events that (a) contain an ErrorDeclaration; and where (b) the error declaration contains a non-null reason field; and where (c) the reason field is equal to one of the values specified in this parameter. If this parameter is omitted, events are returned regardless of whether they contain an ErrorDeclaration or what the value of the reason field is.

Parameter name	Parameter value type	Meaning
`EQ_correctiveEventID`	List of URIs	If this parameter is specified, the result will only include events that (a) contain an `ErrorDeclaration`; and where (b) one of the elements of the `correctiveEventIDs` list is equal to one of the values specified in this parameter. If this parameter is omitted, events are returned regardless of whether they contain an `ErrorDeclaration` or the contents of the `correctiveEventIDs` list.
`EQ_ERROR_DECLARATION_`*`fieldname`*	List of String	Analogous to `EQ_fieldname`, but matches events containing an `ErrorDeclaration` and where the `ErrorDeclaration` contains a field having the specified `fieldname` whose value matches one of the specified values.
`EQ_ERROR_DECLARATION_`*`fieldname`*	Int Float DateTimeStamp Double	Like `EQ_ERROR_DECLARATION_`*`fieldname`* as described above, but may be applied to a field of type Int, Float, Double or Time. The result will include events whose `ErrorDeclaration` (a) has a field named `fieldname`; and where (b) the type of the field matches the type of this parameter (Int, Float, Double or Time); and where (c) the value of the field is equal to the specified value. `fieldname` is constructed as for `EQ_ERROR_DECLARATION_fieldname`.
`GT_ERROR_DECLARATION_`*`fieldname`* `GE_ERROR_DECLARATION_`*`fieldname`* `LT_ERROR_DECLARATION_`*`fieldname`* `LE_ERROR_DECLARATION_`*`fieldname`*	Int Float DateTimeStamp Double	Analogous to `EQ_fieldname`, `GT_fieldname`, `GE_fieldname`, `GE_fieldname`, `LT_fieldname`, and `LE_fieldname`, respectively, but matches events containing an `ErrorDeclaration` and where the `ErrorDeclaration` contains a field having the specified `fieldname` whose integer, float, double-precision floating point or time value matches the specified value according to the specified relational operator.

Parameter name	Parameter value type	Meaning
EQ_INNER_ERROR_DECLARATION_*fieldname*	List of String	Analogous to EQ_ERROR_DECLARATION_fieldname, but matches inner extension elements; that is, any XML field nested within a top-level extension element. Note that a matching inner field may exist within more than one top-level element or may occur more than once within a single top-level element; this parameter matches if at least one matching occurrence is found anywhere in the event (except at top-level).
EQ_INNER_ERROR_DECLARATION_*fieldname*	Int Float DateTimeStamp Double	Like EQ_INNER_ERROR_DECLARATION_*fieldname* as described above, but may be applied to a field of type Int, Float, Double or Time. The result will include events whose inner extension element ErrorDeclaration (a) has a field named fieldname; and where (b) the type of the field matches the type of this parameter (Int, Float, Double or Time); and where (c) the value of the field is equal to the specified value. fieldname is constructed as for EQ_INNER_ERROR_DECLARATION_fieldname.
GT_INNER_ERROR_DECLARATION_*fieldname* GE_INNER_ERROR_DECLARATION_*fieldname* LT_INNER_ERROR_DECLARATION_*fieldname* LE_INNER_ERROR_DECLARATION_*fieldname*	Int Float DateTimeStamp Double	Like EQ_INNER_ERROR_DECLARATION_fieldname as described above, but may be applied to a field of type Int, Float, Double or Time.
EXISTS_ERROR_DECLARATION_*fieldname*	Void	Like EXISTS_fieldname as described above, but events that have an error declaration containing a non-empty extension field named fieldname. Fieldname is constructed as for EQ_ERROR_DECLARATION_fieldname. Note that the value for this query parameter is ignored
EXISTS_INNER_ERROR_DECLARATION_*fieldname*	Void	Like EXISTS_ERROR_DECLARATION_fieldname as described above, but includes events that have an error declaration containing a non-empty inner extension field named *fieldname*. Note that the value for this query parameter is ignored.

Parameter name	Parameter value type	Meaning
`orderBy`	String	If specified, names a single field that will be used to order the results. The `orderDirection` field specifies whether the ordering is in ascending sequence or descending sequence. Events included in the result that lack the specified field altogether may occur in any position within the result event list. The value of this parameter SHALL be one of: `eventTime`, `recordTime`, or the fully qualified name of an extension field whose type is Int, Float, Double, Time, or String. A fully qualified fieldname is constructed as for the `EQ_fieldname` parameter. In the case of a field of type String, sorting SHALL be according to their case-sensitive lexical ordering, considering UTF-8/ASCII code values of each successive character. If omitted, no order is specified. The implementation MAY order the results in any order it chooses, and that order MAY differ even when the same query is executed twice on the same data. (In EPCIS 1.0, the value `quantity` was also permitted, but its use is deprecated in EPCIS 1.1.)
`orderDirection`	String	If specified and `orderBy` is also specified, specifies whether the results are ordered in ascending or descending sequence according to the key specified by `orderBy`. The value of this parameter must be one of `ASC` (for ascending order) or `DESC` (for descending order); if not, the implementation SHALL raise a `QueryParameterException`. If omitted, defaults to `DESC`.

Parameter name	Parameter value type	Meaning
`eventCountLimit`	Int	If specified, the results will only include the first N events that match the other criteria, where N is the value of this parameter. The ordering specified by the `orderBy` and `orderDirection` parameters determine the meaning of "first" for this purpose. If omitted, all events matching the specified criteria will be included in the results. This parameter and `maxEventCount` are mutually exclusive; if both are specified, a `QueryParameterException` SHALL be raised. This parameter may only be used when `orderBy` is specified; if `orderBy` is omitted and `eventCountLimit` is specified, a `QueryParameterException` SHALL be raised. This parameter differs from `maxEventCount` in that this parameter limits the amount of data returned, whereas `maxEventCount` causes an exception to be thrown if the limit is exceeded. Explanation (non-normative): A common use of the `orderBy`, `orderDirection`, and `eventCountLimit` parameters is for extremal queries. For example, to select the most recent event matching some criteria, the query would include parameters that select events matching the desired criteria, and set `orderBy` to `eventTime`, `orderDirection` to `DESC`, and `eventCountLimit` to one.

Parameter name	Parameter value type	Meaning
`maxEventCount`	Int	If specified, at most this many events will be included in the query result. If the query would otherwise return more than this number of events, a `QueryTooLargeException` SHALL be raised instead of a normal query result. This parameter and `eventCountLimit` are mutually exclusive; if both are specified, a `QueryParameterException` SHALL be raised. If this parameter is omitted, any number of events may be included in the query result. Note, however, that the EPCIS implementation is free to raise a `QueryTooLargeException` regardless of the setting of this parameter (see section 8.2.3).
`GE_startTime`	DateTimeStamp	If specified, only events with `startTime` greater than or equal to the specified value will be included in the result. If omitted, events are included regardless of their `startTime` (unless constrained by the `LT_startTime` parameter).
`LT_startTime`	DateTimeStamp	If specified, only events with `startTime` less than the specified value will be included in the result. If omitted, events are included regardless of their `startTime` (unless constrained by the `GE_startTime` parameter).
`GE_endTime`	DateTimeStamp	If specified, only events with `endTime` greater than or equal to the specified value will be included in the result. If omitted, events are included regardless of their `endTime` (unless constrained by the `LT_endTime` parameter).
`LT_endTime`	DateTimeStamp	If specified, only events with `endTime` less than the specified value will be included in the result. If omitted, events are included regardless of their `endTime` (unless constrained by the `GE_startTime` parameter).

Parameter name	Parameter value type	Meaning
EQ_type	List of URIs	If this parameter is specified, the result will only include events that (a) accommodate one or more sensorElement fields; and where (b) the type attribute in one of these sensorElement fields is equal to one of the values specified in this parameter. If this parameter is omitted, events are returned regardless of the value of the type attribute or whether a sensorElement field exists at all.
EQ_deviceID	List of URIs	If this parameter is specified, the result will only include events that (a) accommodate a deviceID attribute; and where (b) the deviceID attribute is equal to one of the URIs specified in this parameter. If this parameter is omitted, events are returned regardless of the value of the deviceID attribute or whether the deviceID attribute exists at all.
EQ_dataProcessingMethod	List of URIs	If this parameter is specified, the result will only include events that (a) accommodate a dataProcessingMethod attribute; and where (b) the dataProcessingMethod attribute is equal to one of the URIs specified in this parameter. If this parameter is omitted, events are returned regardless of the value of the dataProcessingMethod attribute or whether the dataProcessingMethod attribute exists at all.
EQ_microorganism	List of URIs	If this parameter is specified, the result will only include events that (a) accommodate a microorganism attribute; and where (b) the microorganism attribute is equal to one of the URIs specified in this parameter. If this parameter is omitted, events are returned regardless of the value of the microorganism attribute or whether the microorganism attribute exists at all.

Parameter name	Parameter value type	Meaning
`EQ_chemicalSubstance`	List of URIs	If this parameter is specified, the result will only include events that (a) accommodate a `chemicalSubstance` attribute; and where (b) the `chemicalSubstance` attribute is equal to one of the URIs specified in this parameter. If this parameter is omitted, events are returned regardless of the value of the `chemicalSubstance` attribute or whether the `chemicalSubstance` attribute exists at all.
`EQ_bizRules`	List of URIs	If this parameter is specified, the result will only include events that (a) accommodate a `bizRules` attribute; and where (b) the `bizRules` attribute is equal to one of the values specified in this parameter. If this parameter is omitted, events are returned regardless of the value of the `bizRules` attribute or whether the `bizRules` attribute exists at all.
`EQ_value_`*`uom`*	List of Double	If this parameter is specified, the result will only include events that (a) have a `uom` and a `value` attribute; and where (b) a pair of `uom` and `value` is equal or – in case the query includes a `uom` value that is different from those in the events – corresponds to the specified parameter. If omitted, events are in included regardless of the values of their `uom` and `value` attributes.
`GT_value_`*`uom`* `GE_value_`*`uom`* `LT_value_`*`uom`* `LE_value_`*`uom`*	Double	Analogous to `EQ_uom_value`, but includes events whose `uom-value`-pair (or a corresponding `uom-value`-pair when an alternative `uom` is applied) matches the specified relational operator; GT = greater than (>), GE = greater than or equal to (>=), LT = less than (<), LE = less than or equal to (<=).
`GT_minValue_`*`uom`*	Double	Like `GT_value_`*`uom`*, but pertaining to the `minValue` attribute.
`GE_minValue_`*`uom`*	Double	Like `GE_value_`*`uom`*, but pertaining to the `minValue` attribute.
`LT_minValue_`*`uom`*	Double	Like `LT_value_`*`uom`*, but pertaining to the `minValue` attribute.
`LE_minValue_`*`uom`*	Double	Like `LE_value_`*`uom`*, but pertaining to the `minValue` attribute.

Parameter name	Parameter value type	Meaning
GE_maxValue_*uom*	Double	Like GE_value_*uom*, but pertaining to the maxValue attribute.
LT_maxValue_*uom*	Double	Like LT_value_*uom*, but pertaining to the maxValue attribute.
GT_meanValue_*uom*	Double	Like GT_value_*uom*, but pertaining to the meanValue attribute.
GE_meanValue_*uom*	Double	Like GE_value_*uom*, but pertaining to the meanValue attribute.
LT_meanValue_*uom*	Double	Like LT_value_*uom*, but pertaining to the meanValue attribute.
LE_meanValue_*uom*	Double	Like LE_value_*uom*, but pertaining to the meanValue attribute.
GT_sDev_*uom*	Double	Like GT_value_*uom*, but pertaining to the sDev attribute.
GE_sDev_*uom*	Double	Like GE_value_*uom*, but pertaining to the sDev attribute.
LT_sDev_*uom*	Double	Like LT_value_*uom*, but pertaining to the sDev attribute.
LE_sDev_*uom*	Double	Like LE_value_*uom*, but pertaining to the sDev attribute.
EQ_stringValue	List of String	If this parameter is specified, the result will only include events that (a) accommodate a stringValue attribute; and where (b) the stringValue attribute is equal to one of the specified parameter. If this parameter is omitted, events are returned regardless of the value of the stringValue attribute or whether the stringValue attribute exists at all.
EQ_booleanValue	Boolean	If this parameter is specified, the result will only include events that (a) accommodate a booleanValue attribute; and where (b) the booleanValue attribute is equal to the specified value (i.e. 'true' or 'false'). If this parameter is omitted, events are returned regardless of the value of the booleanValue attribute or whether the booleanValue attribute exists at all.

Parameter name	Parameter value type	Meaning
`EQ_hexBinaryValue`	List of String	If this parameter is specified, the result will only include events that (a) accommodate a `hexBinaryValue` attribute; and where (b) the `hexBinaryValue` attribute is equal to one of the values specified in this parameter. If this parameter is omitted, events are returned regardless of the value of the `hexBinaryValue` attribute or whether the `hexBinaryValue` attribute exists at all.
`EQ_uriValue`	List of URIs	If this parameter is specified, the result will only include events that (a) accommodate a `uriValue` attribute; and where (b) the `uriValue` attribute is equal to one of the URIs specified in this parameter. If this parameter is omitted, events are returned regardless of the value of the `uriValue` attribute or whether the `uriValue` attribute exists at all.
`EQ_SENSORELEMENT_`*`fieldname`*	List of String	Analogous to `EQ_fieldname`, but matches events containing a `SensorElement` and where the `SensorElement` contains a field having the specified `fieldname` whose value matches one of the specified values.
`EQ_SENSORELEMENT_`*`fieldname`*	Int DateTimeStamp Double	Like `EQ_SENSORELEMENT_`*`fieldname`* as described above, but may be applied to a field of type Int, Double or Time. The result will include events whose `SensorElement` (a) has a field named `fieldname`; and where (b) the type of the field matches the type of this parameter (integer, double-precision or time); and where (c) the value of the field is equal to the specified value. `fieldname` is constructed as for `EQ_SENSORELEMENT_fieldname`.

Parameter name	Parameter value type	Meaning
`GT_SENSORELEMENT_`*`fieldname`* `GE_SENSORELEMENT_`*`fieldname`* `LT_SENSORELEMENT_`*`fieldname`* `LE_SENSORELEMENT_`*`fieldname`*	Int DateTimeStamp Double	Analogous to `EQ_fieldname`, `GT_fieldname`, `GE_fieldname`, `GE_fieldname`, `LT_fieldname`, and `LE_fieldname`, respectively, but matches events containing a `SensorElement` and where the `SensorElement` contains a field having the specified `fieldname` whose integer, double-precision, or time value matches the specified value according to the specified relational operator.
`EQ_INNER_SENSORELEMENT_`*`fieldname`*	List of String	Analogous to `EQ_SENSORELEMENT_fieldname`, but matches **inner extension elements** (i.e., any XML field nested at any level within a top-level extension element) containing a `SensorElement` and where the `SensorElement` contains a field having the specified `fieldname` whose value matches one of the specified values.
`EQ_INNER_SENSORELEMENT_`*`fieldname`*	Int DateTimeStamp Double	Like `EQ_INNER_ERROR_SENSORELEMENT_`*`fieldname`* as described above, but may be applied to a field of type Int, Double or Time. The result will include events whose inner extension `SensorElement` (a) has a field named `fieldname`; and where (b) the type of the field matches the type of this parameter (integer, double-precision or time); and where (c) the value of the field is equal to the specified value. `fieldname` is constructed as for `EQ_INNER_SENSORELEMENT_fieldname`.
`GT_INNER_SENSORELEMENT_`*`fieldname`* `GE_INNER_SENSORELEMENT_`*`fieldname`* `LT_INNER_SENSORELEMENT_`*`fieldname`* `LE_INNER_SENSORELEMENT_`*`fieldname`*	Int DateTimeStamp Double	Analogous to `EQ_fieldname`, `GT_fieldname`, `GE_fieldname`, `GE_fieldname`, `LT_fieldname`, and `LE_fieldname`, respectively, but matches **inner extension elements** (i.e., any XML field nested at any level within a top-level extension element) containing a `SensorElement` and where the `SensorElement` contains a field having the specified `fieldname` whose integer, double-precision, or time value matches the specified value according to the specified relational operator.

Parameter name	Parameter value type	Meaning
EQ_SENSORMETADATA_*fieldname*	List of String	Analogous to EQ_fieldname, but matches events containing a SensorMetadata element and where the SensorMetadata element contains a field having the specified fieldname whose value matches one of the specified values.
EQ_SENSORMETADATA_*fieldname*	Int DateTimeStamp Double	Like EQ_SENSORMETADATA_*fieldname* as described above, but may be applied to a field of type Int, Double or Time. The result will include events whose SensorMetadata element (a) has a field named fieldname; and where (b) the type of the field matches the type of this parameter (integer, double-precision or time); and where (c) the value of the field is equal to the specified value. fieldname is constructed as for EQ_SENSORMETADATA_fieldname.
GT_SENSORMETADATA_*fieldname* GE_SENSORMETADATA_*fieldname* LT_SENSORMETADATA_*fieldname* LE_SENSORMETADATA_*fieldname*	Int DateTimeStamp Double	Analogous to EQ_fieldname, GT_fieldname, GE_fieldname, GE_fieldname, LT_fieldname, and LE_fieldname, respectively, but matches events containing a SensorMetadata element and where the SensorMetadata element contains a field having the specified fieldname whose integer, double-precision, or time value matches the specified value according to the specified relational operator.
EQ_INNER_SENSORMETADATA_*fieldname*	List of String	Analogous to EQ_fieldname, but matches **inner extension elements** (i.e., any XML field nested at any level within a top-level extension element) containing a SensorMetadata element and where the SensorMetadata element contains a field having the specified fieldname whose value matches one of the specified values.

Parameter name	Parameter value type	Meaning
EQ_INNER_SENSORMETADATA_*fieldname*	Int DateTimeStamp Double	Like EQ_INNER_SENSORMETADATA_*fieldname* as described above, but may be applied to a field of type Int, Double or Time. The result will include events whose inner extension SensorMetadata element (a) has a field named fieldname; and where (b) the type of the field matches the type of this parameter (integer, double-precision or time); and where (c) the value of the field is equal to the specified value. fieldname is constructed as for EQ_INNER_SENSORMETADATA_fieldname.
GT_INNER_SENSORMETADATA_*fieldname* GE_INNER_SENSORMETADATA_*fieldname* LT_INNER_SENSORMETADATA_*fieldname* LE_INNER_SENSORMETADATA_*fieldname*	Int DateTimeStamp Double	Analogous to EQ_fieldname, GT_fieldname, GE_fieldname, GE_fieldname, LT_fieldname, and LE_fieldname, respectively, but matches **inner extension elements** (i.e., any XML field nested at any level within a top-level extension element)containing a SensorMetadata element and where the SensorMetadata element contains a field having the specified fieldname whose integer, double-precision, or time value matches the specified value according to the specified relational operator.
EQ_SENSORREPORT_*fieldname*	List of String	Analogous to EQ_fieldname, but matches events containing a SensorReport element and where the SensorReport element contains a field having the specified fieldname whose value matches one of the specified values.
EQ_SENSORREPORT_*fieldname*	Int DateTimeStamp Double	Like EQ_SENSORREPORT_*fieldname* as described above, but may be applied to a field of type Int, Double or Time. The result will include events whose SensorReport element (a) has a field named fieldname; and where (b) the type of the field matches the type of this parameter (integer, double-precision or time); and where (c) the value of the field is equal to the specified value. fieldname is constructed as for EQ_SENSORREPORT_fieldname.

Parameter name	Parameter value type	Meaning
GT_SENSORREPORT_*fieldname* GE_SENSORREPORT_*fieldname* LT_SENSORREPORT_*fieldname* LE_SENSORREPORT_*fieldname*	Int DateTimeStamp Double	Analogous to EQ_fieldname, GT_fieldname, GE_fieldname, GE_fieldname, LT_fieldname, and LE_fieldname, respectively, but matches events containing a SensorReport element and where the SensorReport element contains a field having the specified fieldname whose integer, double-precision, or time value matches the specified value according to the specified relational operator.
EQ_INNER_SENSOREPORT_*fieldname*	List of String	Analogous to EQ_SENSOREPORT_fieldname, but matches **inner extension elements** (i.e., any XML field nested at any level within a top-level extension element) containing a SensorReport and where the SensorReportcontains a field having the specified fieldname whose value matches one of the specified values.
EQ_INNER_SENSOREPORT_*fieldname*	Int DateTimeStamp Double	Like EQ_INNER_SENSORREPORT_*fieldname* as described above, but may be applied to a field of type Int, Double or Time. The result will include events whose inner extension SensorReport element (a) has a field named fieldname; and where (b) the type of the field matches the type of this parameter (integer, double-precision or time); and where (c) the value of the field is equal to the specified value. fieldname is constructed as for EQ_INNER_SENSORREPORT_fieldname.
GT_INNER_SENSOREPORT_*fieldname* GE_INNER_SENSOREPORT_*fieldname* LT_INNER_SENSOREPORT_*fieldname* LE_INNER_SENSOREPORT_*fieldname*	Int DateTimeStamp Double	Analogous to EQ_fieldname, GT_fieldname, GE_fieldname, GE_fieldname, LT_fieldname, and LE_fieldname, respectively, but matches **inner extension elements** (i.e., any XML field nested at any level within a top-level extension element) containing a SensorElement and where the SensorReport contains a field having the specified fieldname whose integer, double-precision, or time value matches the specified value according to the specified relational operator.

Parameter name	Parameter value type	Meaning
EXISTS_SENSORELEMENT_*fieldname*	Void	Like EXISTS_fieldname as described above, but events that have a SensorElement containing a non-empty extension field named fieldname. Fieldname is constructed as for EQ_SENSORELEMENT_fieldname. Note that the value for this query parameter is ignored.
EXISTS_SENSORMETADATA_*fieldname*	Void	Like EXISTS_fieldname as described above, but events that have a SensorMetadata element containing a non-empty extension field named fieldname. Fieldname is constructed as for EQ_SENSORMETADATA_fieldname. Note that the value for this query parameter is ignored.
EXISTS_SENSORREPORT_*fieldname*	Void	Like EXISTS_fieldname as described above, but events that have a SensorReport element containing a non-empty extension field named fieldname. Fieldname is constructed as for EQ_SENSORREPORT_fieldname. Note that the value for this query parameter is ignored.
GE_percRank	Double	If this parameter is specified, the result will only include events that (a) have a percRank attribute; and where (b) a value of percRank is greater than or equal to the specified parameter. NOTE: since percRank and percValue should be specified together in the data, if present, it may be appropriate for EPCIS queries to express a query constraint of both percRank and percValue together (i.e., not independently of each other).
LT_percRank	Double	If this parameter is specified, the result will only include events that (a) have a percRank attribute; and where (b) a value of percRank is less than the specified parameter. NOTE: since percRank and percValue should be specified together in the data, if present, it may be appropriate for EPCIS queries to express a query constraint of both percRank and percValue together (i.e., not independently of each other).

Parameter name	Parameter value type	Meaning
GE_percValue_uom	Double	If this parameter is specified, the result will only include events that (a) have a percValue attribute; and where (b) a value of percValue is greater than or equal to the specified parameter. NOTE: since percRank and percValue should be specified together in the data, if present, it may be appropriate for EPCIS queries to express a query constraint of both percRank and percValue together (i.e., not independently of each other).
GT_percValue_uom	Double	If this parameter is specified, the result will only include events that (a) have a percValue attribute; and where (b) a value of percValue is greater than the specified parameter. NOTE: since percRank and percValue should be specified together in the data, if present, it may be appropriate for EPCIS queries to express a query constraint of both percRank and percValue together (i.e., not independently of each other).
LE_percValue_uom	Double	If this parameter is specified, the result will only include events that (a) have a percValue attribute; and where (b) a value of percValue is less than or equal to the specified parameter. NOTE: since percRank and percValue should be specified together in the data, if present, it may be appropriate for EPCIS queries to express a query constraint of both percRank and percValue together (i.e., not independently of each other).
LT_percValue_uom	Double	If this parameter is specified, the result will only include events that (a) have a percValue attribute; and where (b) a value of percValue is less than the specified parameter. NOTE: since percRank and percValue should be specified together in the data, if present, it may be appropriate for EPCIS queries to express a query constraint of both percRank and percValue together (i.e., not independently of each other).

As the descriptions above suggest, if multiple parameters are specified an event must satisfy all criteria in order to be included in the result set. In other words, if each parameter is considered to be a predicate, all such predicates are implicitly conjoined as though by an AND operator. For example, if a given call to poll specifies a value for both the EQ_bizStep and EQ_disposition parameters, then an event must

match one of the specified `bizStep` values AND match one of the specified `disposition` values in order to be included in the result.

On the other hand, for those parameters whose value is a list, an event must match *at least one* of the elements of the list in order to be included in the result set. In other words, if each element of the list is considered to be a predicate, all such predicates for a given list are implicitly disjoined as though by an OR operator. For example, if the value of the `EQ_bizStep` parameter is a two-element list ("`bs1`", "`bs2`"), then an event is included if its `bizStep` field contains the value `bs1` OR its `bizStep` field contains the value `bs2`.

As another example, if the value of the `EQ_bizStep` parameter is a two-element list ("`bs1`", "`bs2`") and the `EQ_disposition` parameter is a two-element list ("`d1`", "`d2`"), then the effect is to include events satisfying the following predicate:

((`bizStep` = "bs1" `OR bizStep` = "bs2")
`AND` (`disposition` = "d1" `OR disposition` = "d2"))

8.2.7.1.1 Processing of MATCH query parameters

The parameter list for `MATCH_epc`, `MATCH_parentID`, `MATCH_inputEPC`, `MATCH_outputEPC`, and `MATCH_anyEPC` SHALL be processed as follows. Each element of the parameter list may be a pure identity pattern as specified in [*TDS*], or any other URI. If the element is a pure identity pattern, it is matched against event field values using the procedure for matching identity patterns specified in [*TDS1.9*]. If the element is any other URI, it is matched against event field values by testing string equality.

The parameter list for `MATCH_epcClass`, `MATCH_inputEPCClass`, `MATCH_outputEPCClass`, and `MATCH_anyEPCClass` SHALL be processed as follows. Let *P* be one of the patterns specified in the value for this parameter, and let *C* be the value of an `epcClass` field in the appropriate quantity list of an event being considered for inclusion in the result. Then the event is included if each component *Pi* of *P* matches the corresponding component *Ci* of *C*, where "matches" is as defined in [*TDS*].

Non-Normative: Explanation: The difference between MATCH_epcClass and MATCH_epc, and similar parameters, is that for MATCH_epcClass the value in the event (the epcClass field in a quantity list) may itself be a Pure Identity EPC Pattern URI, as specified in TDS). This means that the value in the event may contain a '*' component. The above specification says that a '*' in the EPCClass field of an event is only matched by a '*' in the query parameter. For example, if the epcClass field within an event is urn:epc:idpat:sgtin:9521321.112345.*, then this event would be matched by the query parameter urn:epc:idpat:sgtin:9521321.*.* or by urn:epc:idpat:sgtin:9521321.112345.*, but not by urn:epc:idpat:sgtin:9521321.112345.400.

8.2.7.2 SimpleMasterDataQuery - REMOVED in EPCIS 2.0

8.2.8 Query callback interface

The Query Callback Interface is the path by which an EPCIS service delivers standing query results to a client.

```
<<interface>>
EPCISQueryCallbackInterface
---
callbackResults(resultData : QueryResults) : void
callbackQueryTooLargeException(e : QueryTooLargeException) : void
callbackImplementationException(e : ImplementationException) : void
```

Each time the EPCIS service executes a standing query according to the `QuerySchedule`, it SHALL attempt to deliver results to the subscriber by invoking one of the three methods of the Query Callback Interface. If the query executed normally, the EPCIS service SHALL invoke the `callbackResults` method. If the query resulted in a `QueryTooLargeException` or `ImplementationException`, the EPCIS service SHALL invoke the corresponding method of the Query Callback Interface.

Note that "exceptions" in the Query Callback Interface are not exceptions in the usual sense of an API exception, because they are not raised as a consequence of a client invoking a method. Instead, the exception is delivered to the recipient in a similar manner to a normal result, as an argument to an interface method.

9 XML bindings for data definition modules

This section specifies a standard XML binding for the Core Event Types data definition module, using the W3C XML Schema language [XSD1, XSD2]. Samples are also shown.

The schema below conforms to GS1 standard schema design rules. The schema below imports the EPCglobal standard base schema, as mandated by the design rules [XMLDR].

9.1 Extensibility mechanism

The XML schema in this section implements the `<<extension point>>` given in the UML of section 7 using a methodology described in [XMLVersioning]. This methodology provides for both vendor/user extension, and for extension by GS1 in future versions of this specification or in supplemental specifications. Extensions introduced through this mechanism will be *backward compatible*, in that documents conforming to older versions of the schema will also conform to newer versions of the standard schema and to schema containing vendor-specific extensions. Extensions will also be *forward compatible*, in that documents that contain vendor/user extensions or that conform to newer versions of the standard schema will also conform to older versions of the schema.

When a document contains extensions (vendor/user-specific or standardised in newer versions of schema), it may conform to more than one schema. For example, a document containing vendor extensions to the GS1 Version 1.0 schema will conform both to the GS1 Version 1.0 schema and to a vendor-specific schema that includes the vendor extensions. In this example, when the document is parsed using the standard schema there will be no validation of the extension elements and attributes, but when the document is parsed using the vendor-specific schema the extensions will be validated. Similarly, a document containing new features introduced in the GS1 Version 1.2 schema will conform to the GS1 Version 1.0 schema, the GS1 Version 1.1 schema, and the GS1 Version 1.2 schema, but validation of the new features will only be available using the Version 1.2 schema.

The design rules for this extensibility pattern are given in [XMLVersioning]. In summary, it amounts to the following rules:

- For each type in which `<<extension point>>` occurs, include an `xsd:anyAttribute` declaration. This declaration provides for the addition of new XML attributes, either in subsequent versions of the standard schema or in vendor/user-specific schema.
- For each type in which `<<extension point>>` occurs, include an optional (`minOccurs` = 0) element named `extension`. The type declared for the `extension` element will always be as follows:

```
<xsd:sequence>
  <xsd:any processContents="lax" minOccurs="1" maxOccurs="unbounded"
           namespace="##local"/>
</xsd:sequence>
<xsd:anyAttribute processContents="lax"/>
```

This declaration provides for forward-compatibility with new elements introduced into subsequent versions of the standard schema.

- For each type in which <<extension point>> occurs, include at the end of the element list a declaration

```
<xsd:any processContents="lax" minOccurs="0" maxOccurs="unbounded"
         namespace="##other"/>
```

  This declaration provides for forward-compatibility with new elements introduced in vendor/user-specific schema.

The rules for adding vendor/user-specific extensions to the schema are as follows:

- Vendor/user-specific attributes may be added to any type in which <<extension point>> occurs. Vendor/user-specific attributes SHALL NOT be in the EPCglobal EPCIS namespace (urn:epcglobal:epcis:xsd:2) nor in the empty namespace. Vendor/user-specific attributes SHALL be in a namespace whose namespace URI has the vendor as the owning authority. (In schema parlance, this means that all vendor/user-specific attributes must have qualified as their form.) For example, the namespace URI may be an HTTP URL whose authority portion is a domain name owned by the vendor/user, a URN having a URN namespace identifier issued to the vendor/user by IANA, an OID URN whose initial path is a Private Enterprise Number assigned to the vendor/user, etc. Declarations of vendor/user-specific attributes SHALL specify use="optional".
- Vendor/user-specific elements may be added to any type in which <<extension point>> occurs. Vendor/user-specific elements SHALL NOT be in the EPCglobal EPCIS namespace (urn:epcglobal:epcis:xsd:2) nor in the empty namespace. Vendor/user-specific elements SHALL be in a namespace whose namespace URI has the vendor/user as the owning authority (as described above). (In schema parlance, this means that all vendor/user-specific elements must have qualified as their form.)

To create a schema that contains vendor/user extensions, replace the <xsd:any … namespace="##other"/> declaration with a content group reference to a group defined in the vendor/user namespace: e.g., <xsd:group ref="vendor:VendorExtension">. In the schema file defining elements for the vendor/user namespace, define a content group using a declaration of the following form:

```
<xsd:group name="VendorExtension">
  <xsd:sequence>
    <!--
      Definitions or references to vendor elements
      go here. Each SHALL specify minOccurs="0".
    -->
    <xsd:any processContents="lax"
             minOccurs="0" maxOccurs="unbounded"
             namespace="##other"/>
  </xsd:sequence>
</xsd:group>
```

(In the foregoing illustrations, vendor and VendorExtension may be any strings the vendor/user chooses.)

Non-Normative: Explanation: Because vendor/user-specific elements must be optional, including references to their definitions directly into the EPCIS schema would violate the XML Schema Unique Particle Attribution constraint, because the <xsd:any …> element in the EPCIS schema can also match vendor/user-specific elements. Moving the <xsd:any …> into the vendor/user's schema avoids this problem, because ##other in that schema means "match an element that has a namespace other than the vendor/user's namespace." This does not conflict with standard elements, because the element form default for the standard EPCIS schema is unqualified, and hence the ##other in the vendor/user's schema does not match standard EPCIS elements, either.

Note: The rules for adding attributes or elements to future versions of the GS1 standard schema are as follows:

- Standard attributes may be added to any type in which `<<extension point>>` occurs. Standard attributes SHALL NOT be in any namespace (i.e., SHALL be in the empty namespace), and SHALL NOT conflict with any existing standard attribute name.
- Standard elements may be added to any type in which `<<extension point>>` occurs. New elements are added using the following rules:
  - Find the innermost `extension` element type.
  - Replace the `<xsd:any … namespace="##local"/>` declaration with (a) new elements (which SHALL NOT be in any namespace; equivalently, which SHALL be in the empty namespace); followed by (b) a new `extension` element whose type is constructed as described before. In subsequent revisions of the standard schema, new standard elements will be added within this new `extension` element rather than within this one.

Non-Normative: Explanation: the reason that new standard attributes and elements are specified above not to be in any namespace is to be consistent with the EPCIS schema's attribute and element form default of `unqualified`.

As applied to the EPCIS 2.0 XML schema for core events (section 9.5), this results in the following:

Event types defined in EPCIS 1.0 appear within the `<EventList>` element.

Event types defined in EPCIS 1.1 (i.e., TransformationEvent) each appear within an `<extension>` element within the `<EventList>` element.

For event types defined in EPCIS 1.0, new fields added in EPCIS 1.1 appear within the `<extension>` element that follows the EPCIS 1.0 fields.

EventType (i.e., `AssociationEvent`) and additional fields (e.g., `sensorElement`) added to EPCIS 2.0 are **not nested in additonal `<extension>` elements**; a set of **XSL transformation tools** published at https://ref.gs1.org/tools/epcis/xsl/ accompanies the EPCIS 2.0 major release, to support implementer migration from EPCIS 1.2 to EPCIS 2.0.

For the `TransformationEvent` (defined in EPCIS 1.1), there is no `<extension>` element, as the entire event type is new in EPCIS 1.1. If additional fields are added in a future version of EPCIS, they will appear within an `<extension>` element following the fields defined in EPCIS 1.1.

Note that the `sensorElement` (EPCIS 2.0) can be used in an `AssociationEvent` (EPCIS 2.0) without embedding, but must be embedded through `<extension>` elements if used in the `TransformationEvent` (EPCIS 1.1).

Vendor/user event-level extensions always appear just before the closing tag for the event (i.e., after any standard fields and any `<extension>` element), and are always in a non-empty XML namespace. Under no circumstances do vendor/user extensions appear within an `<extension>` element; the `<extension>` element is reserved for fields defined in the EPCIS standard itself.

Note: Examples (in XML and JSON/JSON-LD format) are published at https://ref.gs1.org/docs/epcis/examples/.

9.2 Standard business document header

The XML binding for the Core Event Types data definition module includes an optional `EPCISHeader` element, which may be used by industry groups to incorporate additional information required for processing within that industry. The core schema includes a "Standard Business Document Header" (SBDH) as defined in [SBDH] as an optional component of the `EPCISHeader` element. Industry groups MAY also require some other kind of header within the `EPCISHeader` element in addition to the SBDH.

The XSD schema for the Standard Business Document Header may be obtained from the UN/CEFACT website; see [SBDH]. This schema is incorporated herein by reference. GS1's SBDH Technical Implementation Guide [SBDHGS1] provides GS1-specific clarifications around the use of the SBDH.

When the Standard Business Document Header is included, the following values SHALL be used for those elements of the SBDH schema specified below.

SBDH Field (XPath)	Value
`HeaderVersion`	`1.0`
`DocumentIdentification/Standard`	`EPCglobal`
`DocumentIdentification/TypeVersion`	`1.0`
`DocumentIdentification/Type`	As specified below.

The value for `DocumentIdentification/Type` SHALL be set according to the following table, which specifies a value for this field based on the kind of EPCIS document and the context in which it is used.

Document Type and Context	Value for `DocumentIdentification/Type`
`EPCISDocument` used in any context	`Events`
`EPCISQueryDocument` used as the request side of the binding in section 11.3	`QueryControl-Request`
`EPCISQueryDocument` used as the response side of the binding in section 11.3	`QueryControl-Response`
`EPCISQueryDocument` used in any XML binding of the Query Callback interface (section 11.4.2 – 11.4.4)	`QueryCallback`
`EPCISQueryDocument` used in any other context	`Query`

The AS2 binding for the Query Control Interface (section 11.3) also specifies additional Standard Business Document Header fields that must be present in an `EPCISQueryDocument` instance used as a Query Control Interface response message. See section 11.3 for details.

In addition to the fields specified above, the Standard Business Document Header SHALL include all other fields that are required by the SBDH schema, and MAY include additional SBDH fields. In all cases, the values for those fields SHALL be set in accordance with [SBDH]. An industry group MAY specify additional constraints on SBDH contents to be used within that industry group, but such constraints SHALL be consistent with the specifications herein.

9.3 EPCglobal Base schema

The XML binding for the Core Event Types data definition module, as well as other XML bindings in this specification, refer to the EPCglobal Base Schema. This schema is published at https://ref.gs1.org/standards/epcis/2.0.0/epcglobal.xsd.

9.4 Master data in the XML binding

As noted in section 6.1.1, EPCIS provides two ways to transmit master data, supported by different parts of the XML schema specified in the remainder of this section, as summarised in the following table:

Mechanism	Schema Support
ILMD	XML element contained within `ILMD` element
Header of EPCIS document	`VocabularyElement` within `VocabularyList`, as contained within `EPCISHeader`

Each master data attribute is a name/value pair, where the name part is a qualified name consisting of a namespace URI and a local name, and the value is any data type expressible in XML. Regardless of which of the mechanisms above are used to transmit master data, the data transmitted SHALL always use the same namespace URI and local name for a given attribute. The way the namespace URI and local name are encoded into XML, however, differs depending on the mechanism:

- For ILMD elements, the master data attribute SHALL be an XML element whose element name is a qualified name, where the prefix of the qualified name is bound to the namespace URI of the master data attribute and the local name of the qualified name is the local name of the master data attribute. The content of the element SHALL be the value of the master data attribute.
- For the mechanisms that use `VocabularyElement`, the `id` attribute of the `VocabularyElement` element SHALL be a string consisting of the namespace URI, a pound sign (`#`) character, and the local name. The content of the `VocabularyElement` element SHALL be the value of the master data attribute.

Non-Normative: Example: Consider a master data attribute whose namespace URI is `http://epcis.example.com/ns/md`, whose local name is `myAttrName`, and whose value is the string `myAttrValue`. Here is how that attribute would appear in an ILMD section:

```
<epcis:EPCISDocument
    xmlns:epcis="urn:epcglobal:epcis:xsd:2"
    xmlns:example="http://epcis.example.com/ns/md" ...>
  ...
  <ObjectEvent>
    ...
    <ILMD>

<example:myAttrName>myAttrValue</example:myAttrName>
    ...
  </ObjectEvent>
  ...
</epcis:EPCISDocument>
```

And here is how that attribute would appear in a `VocabularyElement`:

```
<VocabularyElement
    id="http://epcis.example.com/ns/md#myAttrName">
  myAttrValue
</VocabularyElement>
```

(Newlines and whitespace have been added on either side of `myAttrValue` for clarity, but they would not be present in actual XML.)

The XML binding for the Core Event Types data definition module includes a facility for the inclusion of additional information in the `readPoint` and `bizLocation` fields of all event types by including additional subelements within those fields following the required `id` subelement. This facility was originally conceived as a means to communicate master data for location identifiers. However, this facility is DEPRECATED as of EPCIS 1.2, and SHOULD NOT be used in EPCIS data conforming to EPCIS 1.2 or later. One or more of the other mechanisms for communicating master data should be used instead.

Vendor extensions (including but not limited to FIT mappings) are tolerated in XML validation by lax processing of the `##other` namespace.

9.5 Schema for core event types

The Core Event Types data definition module, an XML Schema (XSD) artefact, published at https://ref.gs1.org/standards/epcis/2.0.0/epcglobal-epcis-2_0.xsd, imports additional schemas as shown in the following table:

Namespace	Location Reference	Source
`urn:epcglobal:xsd:2`	epcglobal.xsd	section 9.3
`http://www.unece.org/cefact/namespaces/StandardBusinessDocumentHeader`	StandardBusinessDocumentHeader.xsd	UN/CEFACT web site; see section 9.2x

In addition to the constraints implied by the schema, any value of type `xsd:dateTimeStamp` in an instance document SHALL include a time zone specifier (either "`Z`" for UTC or an explicit offset from UTC).

For any XML element that specifies `minOccurs="0"` of type `xsd:anyURI`, `xsd:string`, or a type derived from one of those, an EPCIS implementation SHALL treat an instance having the empty string as its value in exactly the same way as it would if the element were omitted altogether. The same is true for any XML attribute of similar type that specifies `use="optional"`.

This schema also includes the XML binding of master data for the Core Event Types data definition module. The master data portions of the schema are used to provide for an optional master data section of the EPCIS header which may be used in an EPCIS document or EPCIS query document.

The `EPCISDocument` top-level element defined in the schema is used by the concrete bindings of the EPCIS Capture Interface specified in section 11. In addition, trading partners may by mutual agreement use an EPCIS Document as a means to transport a collection of EPCIS events, optionally accompanied by relevant master data, as a single electronic document.

An EPCIS document MAY include master data in its header. This is intended to allow the creator of an EPCIS document to include master data that the recipient of the document might otherwise need to query using the EPCIS Query Interface. It is not required that an EPCIS document include master data in the header, nor is it required that master data in the header include master data for every identifier used in the body of the EPCIS document, or that master data in the header be limited to identifiers used in the body of the EPCIS document. If master data in the header does pertain to an identifier in the body, however, it SHALL be current master data for that identifier at the time the EPCIS document is created. The receiver of an EPCIS document, including an implementation of the EPCIS capture interface, may use or ignore such master data as it sees fit. Master data in the header of an EPCIS document SHALL NOT specify attribute values that conflict with the ILMD section of any event contained within the EPCIS document body.

The XML Schema (XSD) for the Core Event Types data definition module is published at https://ref.gs1.org/standards/epcis/2.0.0/epcglobal-epcis-2_0.xsd.

9.6 Core event types – examples (Non-Normative)

Note: Examples (in XML and JSON/JSON-LD format) are published at https://ref.gs1.org/docs/epcis/examples/.

10 JSON/JSON-LD bindings for data definition

This chapter defines the JSON/JSON-LD data format for EPCIS 2.0. It contains a couple of introductory non-normative sections to provide background and explanation, intended to be helpful for anyone already familiar with the XML data format for EPCIS that is defined in section 9.

Section 10.1 provides a brief introduction to JSON and JSON-LD and why it is considered important that EPCIS 2.0 supports these data formats in addition to XML.

Section 10.2 provides an explanation about how various EPCIS data structures are expressed in XML and JSON/JSON-LD and how these are validated in XSD, JSON Schema and Shape Constraint Language (SHACL) respectively. This section provides a number of examples and is intended to help anyone familiar with the XSD validation of EPCIS data to understand how equivalent validation rules are expressed in JSON Schema and SHACL.

Section 10.3 provides references to the normative validation schema, using JSON Schema to validate the JSON representation and Shape Constraint Language (SHACL) to validate the JSON-LD representation.

Section 10.4 provides references to non-normative examples equivalent to the XML examples provided in section 9.

10.1 Brief introduction to JSON and JSON-LD in the context of EPCIS

EPCIS 2.0 is one of the first GS1 technical standards to support a data format in JSON and JSON-LD as an alternative to XML, which was the only data format specified in EPCIS 1.2 and earlier. XML continues to be supported in EPCIS 2.0.

The additional JSON/JSON-LD data format was motivated by two factors:

1. A desire to support a more lightweight data format that was more familiar to the current generation of software developers
2. A desire to support a Linked Data format for EPCIS data, to enable easier integration of EPCIS data with data from other systems, using Linked Data formats / Resource Description Framework (RDF) as a common framework. (see info box below)

Through the use of scoped contexts within the JSON-LD context file (see section 10.1.3) it has been possible to make the JSON/JSON-LD format even more developer friendly, supporting 'bare word' values for standard CBV code list values for populating `bizStep`, `disposition` or the `type` of a `bizTransaction`, `source`, `destination` or the `type` (measurement type) within the value of `sensorReport`. This means that in JSON/JSON-LD, event data can include key:value pairs such as `"bizStep":"shipping"` or `"disposition":"in_transit"` while the JSON-LD context resource takes care of expanding CBV standard 'bare word' code values such as `"shipping"` or `"in_transit"` to the corresponding Web URIs for standard CBV code values.

About Linked Data

Linked Data is structured data that can be interlinked with other structured data and uses semantic relationships to make factual assertions accessible in a machine-interpretable way. Linked Data is a realization of the vision of the Semantic Web that has gained acceptance by industry.

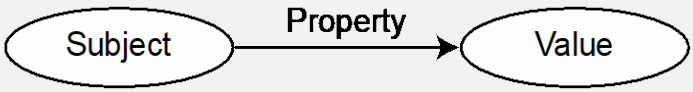
In Linked Data, URIs (ideally Web URIs / IRIs) are used to give each thing a globally unambiguous identifier which can be used to retrieve machine-interpretable facts and also used to express those facts.

The World Wide Web Consortium (W3C) has defined fundamental technical standards for Linked Data / Semantic Web technology, including Resource Description Framework (RDF), RDF Schema (RDFS), Web Ontology Language (OWL), Simple Knowledge Organization System (SKOS) as well as standardised Linked Data formats such as JSON for Linked Data (JSON-LD).

Linked Data can be used to represent facts in a directed data 'graph' or network consisting of nodes (representing things of interest or simple values such as strings, dates or numbers) joined together by directed arcs that corresponds to predicates, properties or relationships.

At the most fundamental level, Linked Data uses the idea of an RDF Triple consisting of a Subject, Property and Value (also referred to as Subject, Predicate and Object).

Figure 10-1 RDF Triple: Subject-Property-Value



Linked Data triples provide a 'lowest common denominator' across various data formats, whether the source data is formatted as tables or spreadsheets or comma-separated values or whether it is more hierarchical, such as XML.

Linked Data technology also includes query languages such as SPARQL that can be used to perform simple or complex semantic queries across multiple distributed datasets as easily as if all the data were present in the same local database.

For this reason, there is some interest in having a Linked Data representation (e.g. JSON-LD format) for EPCIS data, to simplify data integration and queries when EPCIS data is combined / compared with data from other systems.

10.1.1 JavaScript Object Notation (JSON)

JavaScript Object Notation (JSON) is defined in RFC 8259 [https://datatracker.ietf.org/doc/html/rfc8259] and as ISO/IEC 21778:2017 [ISO21778]. JSON provides a lightweight data interchange format that can be used across multiple programming or scripting languages for the exchange of structured data. Compared with XML, JSON is simpler and usually also more compact and better supported in many modern programming languages without the need to include an additional processing library.

JSON supports four simple data types:

- String – a sequence of zero or more Unicode characters enclosed within double quotes
- Number – supports positive or negative integers and floating-point numbers, optionally using exponential E notation, e.g. -1.2345E-6
- Boolean – either `true` or `false`
- `null`

JSON also supports two complex data types:

- Arrays or Lists – a comma-separated sequence of zero or more elements enclosed within square brackets, e.g. [1,"xyz",`true`]
- Objects or Associative Arrays or Dictionaries – a comma-separated sequence of zero or more key:value pairs enclosed within curly brackets, e.g. {"key1":2,"key2":"abc"}

Due to its simplicity, JSON can simplify the expression of some data structures (especially arrays/lists) but it lacks some of the features present in XML:

- XML uses named element tags to declare that everything between the opening and closing tags is of a specific data type (often user-defined). Declarations within XML Schema Definition (XSD) link a named element to a defined data structure.
- XML uses `hreflang` to declare that a string value is provided in a specific human language.

- XML Schema Definition language (XSD) can define that some specific string values should be cast to (interpreted as) other data types, such as specific XSD data types for dates, date+time timestamps etc.

10.1.2 JSON for Linked Data (JSON-LD)

JSON for Linked Data [JSON-LD] is a W3C technical recommendation that defines an extended JSON format that addresses some of these shortcomings of JSON, as well as positioning JSON-LD as another serialisation format for Linked Data (logical triples of data), within the Resource Description Framework (RDF) alongside other Linked Data formats such as [RDFa], Terse Triple Notation [Turtle], [RDF/XML] etc.

Because EPCIS 1.2 already provides an XML data format that makes use of some of these features missing in JSON (explicit data type casting, support for multiple namespaces) and because some use cases were seeking a Linked Data format for EPCIS data, EPCIS 2.0 provides a JSON/JSON-LD data format. What is meant by JSON/JSON-LD is that as far as possible, most annotations that are specific to JSON-LD are hidden from the body of the data payload by placing them within the JSON-LD context resource and through the use of aliases. This means that if the software consuming EPCIS 2.0 data only expects to treat it as JSON data, it can simply ignore the JSON-LD context resource and use existing JSON processing functions for parsing the body of the data payload. Software that specifically needs EPCIS 2.0 as a JSON-LD or Linked Data format makes use of the JSON-LD context resource (including retrieval of any referenced online context files/resources) to obtain a full JSON-LD document / data structure, which can also be easily translated into other Linked Data formats using available translation tools.

For the JSON-LD format, EPCIS 2.0 requires a minimum of JSON-LD v1.1 because it makes use of some features such as protected term definitions (using `"@protected"`) that were not defined in JSON-LD v1.0.

EPCIS 2.0 makes use of the following special keywords of JSON-LD:

Special keyword	Explanation
`"@id"`	`"@id"` is used to specify the Subject of Linked Data triples within data objects of associative arrays; for each key:value pair, the key is interpreted as an RDF Property or Predicate of the Subject that was specified by the value of the special @id key, while the corresponding value is interpreted as the corresponding RDF Object or Value for the same Subject – Predicate – Object triple or Subject – Property – Value triple. For example, the EPCIS field `"eventID"` is declared as `"@id"` to indicate that its value (such as `"ni:///sha-256;c6407ffcac52ec159528f2b556ba4ac3844c5aa48485c1fd61643e94f0a2d678?ver=CBV2.0"`) is an IRI/URI to be treated as the Subject of various RDF triples. `"@id"` is also used to map a JSON key to an IRI/URI for the corresponding Linked Data property. For example, to enable a simpler JSON syntax such as `"bizStep": "shipping"`, `"bizStep"` is declared as `"@id":"epcis:bizStep"`, which in turn expands to `"@id":"https://ref.gs1.org/epcis/bizStep"`.
`"@type"`	`"@type"` is used to specify that a string value should be cast to another data type, such as an xsd:dateTimeStamp.
`"@context"`	`"@context"` is used to specify the JSON-LD context resource, either declared explicitly inline or via URL reference. See section 10.1.3 for further details. The JSON-LD context resources for EPCIS 2.0 and CBV 2.0 make use of scoped contexts (defined locally within an EPCIS property or field) and protected term definitions, in order to support correct mapping of bare string values for property names and standard CBV code values, to enable a much simpler JSON syntax such as `"bizStep": "shipping"` when standard CBV values are used.
`"@protected"`	`"@protected": true` is used to prevent the definition of a term being overridden by other JSON-LD contexts.

Special keyword	Explanation
`"@version"`	`"@version"` is set to 1.1 to ensure that implementations of only JSON-LD v1.0 do not attempt to process the EPCIS 2.0 / CBV 2.0 context resources, which depend on features (such as `"@protected"`) that were introduced in JSON-LD v1.1.
`"@type": "@id"`	When a property is declared to be of `"@type": "@id"`, its value is interpreted (cast) as an IRI/URI rather than a string. EPCIS properties such as `readPoint`, `bizLocation`, `epcList`, `parentID`, `deviceID`, `microorganism` etc. make use of this.
`"@type": "@vocab"`	When a property is declared to be of `"@type": "@vocab"`, the context resource can define a set of standard bare-word strings that can expand to IRIs / URIs for enumerations of standard values, such as CBV standard code lists for populating EPCIS properties such as `bizStep`, `disposition`, `set` or `unset` within `persistentDisposition`, `type` within `bizTransactionList` or `type` within `sensorReport`. This enables a much simpler JSON syntax such as `"bizStep": "shipping"` when standard CBV values are used.
`"@container": "@set"`	When a property is declared with `"@container": "@set"`, its values are treated as an unordered set, rather than an ordered list. EPCIS properties such as `epcList`, `childEPCs`, `inputEPCList` etc. make use of this feature, whereas some other properties such as `parentID` do not declare this because they expect a single value rather than an unordered set of zero or more values.

Many users of EPCIS 2.0 will initially focus on the JSON representation and may disregard the context resource. However, retrieval and usage of the context resource enables the full Linked Data potential of the JSON-LD representation to be reached.

In order to hide such special JSON-LD keywords from the JSON body of the data, EPCIS 2.0 makes use of a JSON-LD context resource to express namespace mappings, aliases and to cast values of particular data fields to specific data types. Features of the JSON-LD context resource that are relevant for EPCIS 2.0 are explained in section 10.1.3.

It should be noted that a JSON-LD dataset may reference or include more than one context resource. Typically, it may reference **the standard JSON-LD context resource for EPCIS 2.0, which is published at https://ref.gs1.org/standards/epcis/2.0.0/epcis-context.jsonld**. It may also reference one or more additional JSON-LD context resources that provide additional namespace mappings and aliases for terms from other namespaces such as those defined by users, industry groups or solution providers.

10.1.3 Features of the JSON-LD context resource

The context resource is a data object container in which the following may be specified:

- Expansions for Compact URI Expression (CURIE) prefixes,
  e.g. `"xsd": "http://www.w3.org/2001/XMLSchema#"`
  In this sense, the context resource is used in a similar way to the way QName namespace declarations are expressed within an XML header.
  - Custom namespace expansions can also be declared within a context resource, e.g. `"example": "http://ns.example.com/epcis/"`
- Mappings of fields to IRIs/URIs of Linked Data properties and their values to data types other than string (type casting), e.g.:
  - `"eventTime": {"@id":"epcis:eventTime", "@type":"xsd:dateTimeStamp"}`
    The JSON key "eventTime" should be mapped to Linked Data property https://ref.gs1.org/epcis/eventTime
    Every value for eventTime (and also recordTime and declarationTime) should be treated as an xsd:dateTimeStamp value
  - `"quantity": {"@id":"epcis:quantity", "@type":"xsd:double"}`
    The JSON key "quantity" should be mapped to Linked Data property

https://ref.gs1.org/epcis/*quantity*
Every value of quantity should be treated as an xsd:double value

- `"epcList": {"@id":"epcis:epcList", "@type":"@id", "@container": "@set"}`
  The JSON key "epcList" should be mapped to Linked Data property https://ref.gs1.org/epcis/epcList
  Every value within epcList (and also epcClass, parentID, childEPCs, inputEPCList, outputEPCList, bizStep, disposition, readPoint, bizLocation, bizTransaction, source, destination, type) should be treated as IRIs –JSON-LD uses the special notation "@id" for this purpose.
  The values are to be treated as an unordered set, rather than an ordered list.

- Aliases for JSON-LD keywords, e.g.:
  - "type": "@type"
  - "id":"@id"
  - Mappings of EPCIS fields to semantic URIs defined elsewhere, e.g.:
  - "creationDate": {"@id":"dcterms:created", "@type":"xsd:dateTime"}
  - "schemaVersion": {"@id":"owl:versionInfo"}
    The values are data objects in which the value for the `"@id"` key is the semantic URI or CURIE to which the EPCIS fieldname corresponds, while the value for the `"@type"` key is the corresponding data type.

- Mappings of sets of 'bare word' values to corresponding Web URIs for standard CBV code list values with properties such as `bizStep`, `disposition` etc. that can be populated with standard CBV code list values. The standard JSON Schema for EPCIS includes the same sets of 'bare word' values as an enumeration and requires that the value of such properties is either a string from such a defined enumeration or 'bare word' values or a URI if it is a custom value from another namespace (e.g. defined by a user, sector, region or solution provider). For example, in the JSON/JSON-LD format for EPCIS, `bizStep` is expected to be either a string from an enumerated list that includes CBV code values for populating `bizStep` (such as `"shipping"`, `"receiving"`, `"commissioning"` etc.) or otherwise it must be a URI if it is not using one of those standard CBV code values.

Even though each individual member object in a JSON-LD document MAY declare its local @context, EPCIS 2.0 capturing applications that use the JSON-LD context resource SHOULD specify a consolidated `@context` for all default and user-defined namespaces at the root level of the EPCISDocument (refers to the payload of POST /capture endpoint) **as well as** at the root level of the EPCISEvent (refers to POST /event endpoint). This is strongly recommended, in order to simplify document parsing and validation.

10.1.4 Compact URI Expressions (CURIEs)

JSON-LD also recognises the use of Compact URI Expressions (CURIEs) [https://www.w3.org/TR/curie/] as a more compact way of expressing Uniform Resource Identifiers (URIs) especially when a document or data structure includes several URI values that share a common stem and structure, only differing in the value of the final URI component.

A CURIE is written as two components joined by a colon in the format `prefix:finalpart`

The prefix (the part before the colon) is a string whose CURIE expansion should be defined in the context resource. To expand the CURIE to the corresponding full URI, the final part is simply appended to the CURIE expansion of the prefix.

For example, the context resource defines a CURIE expansion for "xsd" (XML Schema Datatypes) as follows:

```
"xsd" : "http://www.w3.org/2001/XMLSchema#"
```

This means that a CURIE such as `xsd:dateTimeStamp` should be expanded to http://www.w3.org/2001/XMLSchema#dateTimeStamp

CURIEs are syntactically similar to QNames in XML but whereas a QName provides globally unambiguous namespace qualification for element names and attribute names, CURIEs are designed to always expand to a full URI or Internationalised Resource Identifier (IRI). In a CURIE, the local part following the colon is not required to be a valid XML element name, so for example, an all-numeric local part following the colon is valid in a CURIE even though it would not be valid in a QName. QNames can therefore be considered as a subset of CURIEs.

10.2 Expression and validation of EPCIS data structures in JSON and JSON-LD

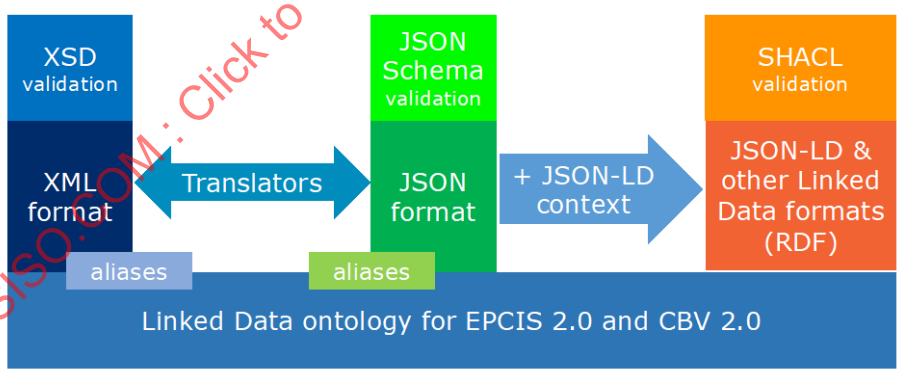
EPCIS v1.2 defines an XML data format which is validated using XML Schema Definition language (XSD). In addition to the XML data format (which is updated in Epcis 2.0 to support the new `AssociationEvent` and `SensorElement` and `persistentDisposition`), Epcis 2.0 defines a JSON/JSON-LD data format.

JSON data can be validated using JSON Schema [https://json-schema.org/specification.html]. EPCIS/CBV 2.0 is compatible with version 7 of JSON Schema and with any future version of JSON Schema that maintains full backward compatibility with JSON Schema version 7, since it makes use of the `if`/`then` feature introduced in JSON Schema v7 and is used to avoid spurious validation results by ensuring that validation rules can be specific to each event type.

JSON-LD data can be validated using Shape Constraint Language (SHACL), a W3C technical recommendation [https://www.w3.org/TR/shacl/]

JSON Schema and SHACL play an analogous role to XML Schema Definition language (XSD) for the validation of XML data. EPCIS 2.0 defines JSON Schema and SHACL files for validation purposes. Some software toolkits may also make use of these for the generation of stub code to reduce development effort and ensure consistency with the schema.

Figure 10-2 Supporting multiple formats for EPCIS / CBV 2.0



The following sections explain how the EPCIS data structures are expressed in XML and JSON/JSON-LD as well as how validation rules are expressed for simple literal values, enumerations, lists/arrays and more complex data structures, providing a side-by-side comparison of the syntax expressed in XSD, JSON Schema and SHACL to help readers of this standard understand how each validation rule is supported in these three validation languages. The following subsections are intended as a non-normative introduction. The normative validation rules are expressed in the XSD, JSON Schema and SHACL files published for EPCIS 2.0.

10.2.1 Expressing data fields expecting simple values

Some EPCIS event fields (such as `action`, `bizStep`, `disposition`, `eventTime`, `recordTime`) each expect a single data value, not a list. In XML, the value appears between the opening and closing tags of an XML element named after the corresponding field.

In JSON/JSON-LD, there are no closing tags and no named elements. Instead, event fields expecting simple values are represented as key:value pairs within a data object that is enclosed within curly brackets.

The tables below show equivalent examples for how such data might be expressed.

Format	EPCIS 2.0 example for action
XML	`<action>OBSERVE</action>`
JSON	`"action": "OBSERVE"`
JSON-LD	Within the standard JSON-LD context resource, `"action": "epcis:action"` (maps to `https://ref.gs1.org/epcis/action`)

Format	EPCIS 2.0 example for eventTime (same pattern for recordTime)
XML	`<eventTime>2005-04-03T20:33:31.116-06:00</eventTime>`
JSON	`"eventTime": "2005-04-03T20:33:31.116000-06:00"`
JSON-LD	Within the standard JSON-LD context resource, `"eventTime": {"@id":"epcis:eventTime", "@type":"xsd:dateTimeStamp"}`

Format	EPCIS 2.0 example for bizStep (same pattern for disposition)
XML	`<bizStep>urn:epcglobal:cbv:bizstep:shipping</bizStep>`
JSON	`"bizStep": "shipping"`

JSON-LD	Within the standard JSON-LD context resource, such fields define a protected scoped context that defines expansion of the corresponding standard CBV values. A snippet is shown below, with the ellipsis indicating numerous standard CBV values for populating `bizStep` that are not shown in this snippet.
	
	`"bizStep": {`
	`  "@context": {`
	`  "@protected": true,`
	`  "@version": 1.1,`
	`  "cbv": "https://ref.gs1.org/cbv/",`
	`  "accepting": "cbv:BizStep-accepting",`
	`  ...`
	`  "shipping": "cbv:BizStep-shipping",`
	`  ...`
	`  "void_shipping": "cbv:BizStep-void_shipping"`
	`  },`
	`  "@id": "epcis:bizStep",`
	`  "@type": "@vocab"`
	`  },`

Note that in each of these examples, the JSON data consists of a simple key:value pair within a data object, while the mapping to specific data types is expressed within the JSON-LD context resource. Note also that for `bizStep`, `disposition` and other fields that expect a URI value, the context resource indicates `"@type":"@vocab"` and defines a protected scoped context that defines the mappings from simple string values such as `"shipping"` to the corresponding Web URIs such as `<https://ref.gs1.org/cbv/BizStep-shipping>`.

10.2.2 Validating data fields expecting simple values

For simple values that are not an `xsd:string`, the JSON-LD context resource specifies the `@type` for each field within an EPCIS event. Note that where the simple value is a URI, JSON-LD indicates this via the special key:value pair `"@type":"@id"` or `"@type":"@vocab"` within the JSON-LD context resource, `"@id"` indicating that the value should be considered as an IRI or URI that is a potential Linked Data resource with additional properties of its own, rather than a literal URI; for this reason, EPCIS fields that expect a URI are never mapped using `"@type":"xsd:anyURI"`. The special key:value pair `"@type":"@vocab"` indicates that the value is an IRI/URI from a defined vocabulary and the standard JSON-LD context resource for EPCIS/CBV 2.0 provides mappings from bare strings such as `"shipping"` to standard CBV values (such as `<https://ref.gs1.org/cbv/BizStep-shipping>`) that are appropriate for the respective property or data field in which each scoped context is defined.

The following tables show how the validation rules for two of the above examples are expressed using XSD, JSON Schema and SHACL. Colour coding is used to indicate equivalent ways of expressing constraints. For example, the name of the property is highlighted in red. The expected data type is shown in blue. The purple highlighting shows how each validation format expresses that the property is mandatory. The validation rules for the `action` field are discussed in section 10.2.3.

Format	EPCIS 2.0 validation example for eventTime (similar pattern for recordTime)
XSD	`<xsd:element name="eventTime" type="xsd:dateTimeStamp"` `  minOccurs="1" maxOccurs="1" />`

Format	EPCIS 2.0 validation example for eventTime (similar pattern for recordTime)

JSON Schema	`"definitions": {`
	`  "time": {"type": "string","format": "date-time" }`
	`  …`
	`}`
	`"properties" : [`
	`  "eventTime": {"$ref": "#/definitions/time"},`
	`  …`
	`]`
	`"required" : ["eventTime"]`

SHACL	`epcis:EventTimeShape`
	`  a sh:PropertyShape ;`
	`  sh:path epcis:eventTime ;`
	`  sh:name "eventTime" ;`
	`  sh:datatype xsd:dateTimeStamp ;`
	`  sh:minCount 1 ;`
	`  sh:maxCount 1 ;`
	`  sh:message "In all EPCIS events, eventTime is mandatory, single valued and an xsd:dateTimeStamp" ;`

Because `eventTime` is a mandatory field in all EPCIS events, it must occur exactly once.

For XSD, the default values of both attributes `minOccurs` and `maxOccurs` is 1, so they are typically omitted from XSD validation rules unless they specify other values such as `minOccurs="0"` for an optional field or `maxOccurs="unbounded"` if there is no upper limit on the number of permitted values.

In JSON Schema, a mandatory field is listed within the list of `"required"` fields.

In SHACL, a mandatory field is indicated via a constraint of `sh:minCount 1`.

In XSD, the `type` attribute expresses the data type, e.g. `type="xsd:dateTimeStamp"`.

In JSON Schema, the property for `eventTime` references a definition for a datatype called `time`, in which the `"type"` is required to be `"string"` and the `"format"` is required to be `"date-time"`.

In SHACL, the constraint `sh:datatype` specifies that an `xsd:dateTimeStamp` value is expected.

In the next example, bizStep is an optional field in EPCIS events and a URI value is expected. The corresponding validation rules are shown in the table below.

Format	EPCIS 2.0 validation example for bizStep (similar pattern for disposition)

XSD	`<xsd:element name="bizStep" type="epcis:BusinessStepIDType"`
	`  minOccurs="0"  maxOccurs="1"  />`
	
	`<xsd:simpleType name="BusinessStepIDType">`
	`  <xsd:restriction base="xsd:anyURI"/>`
	`</xsd:simpleType>`

Format	EPCIS 2.0 validation example for bizStep (similar pattern for disposition)

JSON Schema	"definitions": {
	"vocab-uri": {"type": "string","format": "uri" }
	…
	}
	"properties" : [
	"bizStep": {
	"anyOf": [
	{ "$ref": "#/definitions/vocab-uri" },
	{
	"type": "string",
	"enum": [
	"accepting",
	"arriving",
	…
	"unpacking",
	"void_shipping"
	]
	}
	]
	}
	...
	]

SHACL	epcis:BizStepShape a sh:PropertyShape ; sh:path epcis:bizStep ; sh:name "bizStep" ; sh:nodeKind sh:IRI ; sh:pattern "^(.+?):(.+)$" ; sh:maxCount 1; sh:message "bizStep must be single-valued and an IRI/URI" ;

In XSD, `minOccurs="0"` because the `bizStep` field is not mandatory. The `type` attribute specifies that the value is a type `epcis:BusinessStepIDType` which is effectively an `xsd:anyURI` according to its definition.

In JSON Schema, the equivalent to specifying `minOccurs="0"` is to declare the field within the list of `"properties"` but to omit it from the list of `"required"` fields. `bizStep` then supports two alternative formats, indicated by `"anyOf"`. The first alternative references a definition named `vocab-uri`, in which the `"type"` attribute specifies `"string"` and the `"format"` attribute specifies `"uri"`; this is to support values from custom vocabularies outside of the standard CBV values defined for `bizStep`. The second alternative uses `"enum"` to enumerate a set of bare string values (such as `"arriving"`) corresponding to the standard CBV values defined for `bizStep`; the standard JSON-LD context for EPCIS/CBV 2.0 then expands these bare string values to the corresponding URIs such as `<https://ref.gs1.org/cbv/BizStep-arriving>`.

In SHACL, the equivalent to specifying `minOccurs="0"` is to declare a constraint `sh:minCount` of 0, although this is the default value for `sh:minCount`, so it is acceptable to omit `sh:minCount` for optional fields, since the default value (0) is assumed. For mandatory fields, it is necessary to assert `sh:minCount 1` whereas in XSD, `minOccurs="1"` is the default and assumed if the minOccurs attribute is not specified. Note that for fields that expect a URI value, instead of using `sh:datatype`, the validation rule uses `sh:nodeKind sh:IRI`.

10.2.3 Validation of fields (e.g. 'action') that expect a string value from an enumerated list

The `action` field is present within `ObjectEvent`, `AggregationEvent`, `TransactionEvent` and `AssociationEvent` but absent from `TransformationEvent`. Its value is a string from an enumerated list consisting of three options, `"ADD"`, `"OBSERVE"`, `"DELETE"`.

The table below shows how to specify validation rules for the permitted enumerated values of the action field in XSD, JSON Schema and SHACL.

Format	EPCIS 2.0 validation example for action
XSD	<xsd:element name="action" type="epcis:ActionType"/> <xsd:simpleType name="ActionType"> <xsd:restriction base="xsd:string"> <xsd:enumeration value="ADD"/> <xsd:enumeration value="OBSERVE"/> <xsd:enumeration value="DELETE"/> </xsd:restriction>
JSON Schema	"action": { "type": "string", "enum": ["ADD","OBSERVE","DELETE"] },
SHACL	epcis:ActionShape a sh:PropertyShape ; sh:path epcis:action ; sh:name "action" ; sh:datatype xsd:string ; sh:in ("ADD" "OBSERVE" "DELETE") ; sh:minCount 1 ; sh:maxCount 1 ; sh:message "Within ObjectEvent, AggregationEvent, TransactionEvent, AssociationEvent, action is mandatory, and must be a string value, one of either 'ADD', 'OBSERVE' or 'DELETE'" ; .

In XSD, enumerated strings are expressed via `xsd:enumeration` elements within `xsd:simpleType` with an `xsd:restriction` base attribute value of `xsd:string`.

In JSON Schema, the same enumerated strings are the list values of the `"enum"` property.

In SHACL, the same enumerated strings are within the list values of the `"sh:in"` property.

10.2.4 Expressing simple lists of values

Some EPCIS event fields (such as `epcList`, `childEPCs`) expect a list of zero or more values. In XML, each element of the list is enclosed within an `<epc>` or `<id>` element (or the `<set>` and `<unset>` elements within `<persistentDisposition>`), which appear nested within the XML element that expects a list of values, as shown in the table below.

In the JSON/JSON-LD data format, a list is natively supported using the square bracket notation to indicate a list of comma-separated values, so in JSON/JSON-LD, there is usually no need for anything equivalent to the wrapper element for each element of the list. The exception to this is for child elements in a list in which an inline attribute is permitted in XML. This is discussed in section 10.2.6.

Format	EPCIS 2.0 example for epcList
XML	<epcList> <epc>urn:epc:id:sgtin:9521321.007346.2017</epc> <epc>urn:epc:id:sgtin:9521321.007346.2018</epc> </epcList>
JSON	{"epcList": ["urn:epc:id:sgtin:9521321.007346.2017","urn:epc:id:sgtin:9521321.007346.2018"]}
JSON-LD	Within JSON-LD context resource, `"epcList": {"@id":"epcis:epcList", "@type":"@id", "@container":"@set"}` because the EPC values are URIs (indicated by `"@type":"@id"`) within an unorderd list (set) (indicated by `"@container":"@set"`).

10.2.5 Validating lists of values

Format	EPCIS 2.0 validation example for epcList

XSD	`<xsd:element name="epcList" type="epcis:EPCListType"/>`
	
	`<xsd:complexType name="EPCListType">`
	`  <xsd:sequence>`
	`  <xsd:element name="epc" type="epcglobal:EPC" minOccurs="0" maxOccurs="unbounded"/>`
	`  </xsd:sequence>`
	`</xsd:complexType>`

JSON Schema	`"definitions": {`
	`  "uri": {"type": "string","format": "uri" }`
	`  …`
	`}`
	`"properties" : [`
	`  "epcList": {"type": "array", "items": { "$ref": "#/definitions/uri" },`
	`  …`
	`]`

SHACL	`epcis:EPCListShape`
	`  a sh:PropertyShape ;`
	`  sh:path epcis:epcList ;`
	`  sh:name "epcList" ;`
	`  sh:nodeKind sh:IRI ;`
	`  sh:pattern "^(.+?):(.+)$" ;`
	`  sh:message "Any values within epcList must be IRIs/URIs" ;`
	`.`

In JSON Schema, the field is declared to be of `"type":"array"`, while the structure of the elements of the array is specified via the `"items"` keyword, which in this case uses `"$ref"` to reference a definition for a reference named `"uri"` that specifies a value of `"type": "string"` and `"format": "uri"`.

In SHACL, there is no need to declare that a list of values is expected. `sh:nodeKind sh:IRI` indicates that the values are expected to be IRIs or URIs.

10.2.6 Expressing lists of elements with inline attributes expressing type

In XML, `<bizTransactionList>` contains repeated child elements named `<bizTransaction>`, each having an inline type attribute. `<sourceList>` and `<destinationList>` share a similar structure, in which there may be repeated child elements named `<source>` or `<destination>`, each having an optional inline `type` attribute.

In JSON/JSON-LD, `bizTransactionList` expects a list of objects, each having a `bizTransaction` field and a `type` field. There is no concept of an inline attribute within JSON/JSON-LD, so what appear in XML as elements containing bare child string values and inline attributes are treated equally. In JSON/JSON-LD, this results in a JSON data object with two or more key:value pairs. As shown in the example below, one pair (such as `"type" : "po"`) is formed from each XML inline attribute, while the final pair (e.g. `"bizTransaction": "http://transaction.acme.com/po/12345678"`) is formed by setting the key to the name of the XML element that contains a bare child string value (e.g. setting a JSON key of `"bizTransaction"` from the XML element `<bizTransaction>`) and setting its value to the bare child string value that was enclosed within the XML element. Note that for standard CBV code list values, the JSON/JSON-LD format for EPCIS/CBV 2.0 uses bare words such as `"po"`, `"shipping"` etc. and relies upon mappings defined within protected scoped contexts that are appropriate for the corresponding property/field, as explained in section 10.2.1.

Format	EPCIS 2.0 example for bizTransactionList
XML	<bizTransactionList> <bizTransaction type="urn:epcglobal:cbv:btt:po"> http://transaction.acme.com/po/12345678 </bizTransaction> <bizTransaction type="urn:epcglobal:cbv:btt:desadv"> urn:epcglobal:cbv:bt:9521321073467:1152 </bizTransaction> </bizTransactionList>
JSON JSON-LD	{"bizTransactionList": [{"type": "po", "bizTransaction": "http://transaction.acme.com/po/12345678" }, {"type": "desadv", "bizTransaction": "urn:epcglobal:cbv:bt:9521321073467:1152" }]}

Within the standard JSON-LD context resource for EPCIS/CBV 2.0, the following entry appears for "bizTransactionList":

```
"bizTransactionList": {
  "@id": "epcis:bizTransactionList",
  "@container": "@set",
  "@context": [
    {
      "@protected": true,
      "@version": 1.1,
      "cbv": "https://ref.gs1.org/cbv/",
      "bol": "cbv:BTT-bol",
        …
      "desadv": "cbv:BTT-desadv",
        …
      "po": "cbv:BTT-po",
        …
      "upevt": "cbv:BTT-upevt"
    },
    {
      "@protected": true,
      "epcis": "https://ref.gs1.org/epcis/",
      "bizTransaction": {
        "@id": "@id",
        "@type": "@id"
      },
      "type": {
        "@id": "epcis:bizTransactionType",
        "@type": "@vocab"
      }
    }
  ]
}
```

`"@container": "@set"` indicates that the value of `bizTransactionList` is an unordered set of items. A protected "@context" within the scope of `bizTransactionList` defines that the `bizTransaction` field is an IRI/URI (`"@type": "@id"`) and that it represents the Subject of an RDF triple (`"@id": "@id"`), while within the local context of `bizTransactionList` the `type` field is mapped to the property epcis:bizTransactionType (`"@id": "epcis:bizTransactionType"`) and its value is a URI from a controlled vocabulary (`"@type": "@vocab"`) that is enumerated within the same protected scoped context that expresses mappings such as `"desadv": "cbv:BTT-desadv" ( = https://ref.gs1.org/cbv/BTT-desadv )` and `"po": "cbv:BTT-po" ( = https://ref.gs1.org/cbv/BTT-po )`.

Within the ontology file for EPCIS, the Linked Data property `epcis:bizTransactionType` corresponds to `type` within `bizTransactionList`. Similarly, the Linked Data property `epcis:sourceOrDestinationType` corresponds to `source` within `sourceList` and to `destination` within `destinationList`. Likewise, the Linked Data property `epcis:measurementType` corresponds to `type` within `sensorReport`. These JSON labels (`'type','source','destination'`) are noted within the EPCIS ontology via the annotation property `epcis:jsonldLabel` from the properties that use them.

`bizTransaction`, `source` and `destination` are not defined as properties within the EPCIS ontology. Instead, they are considered as JSON aliases of the special "@id" keyword in JSON-LD because their value is a URI that is the RDF Subject of a triple that expresses the `type`, which is mapped via the standard JSON-LD context file for EPCIS to the properties `epcis:bizTransactionType` or `epcis:sourceOrDestinationType` depending on the scope in which they are used. That is why the standard JSON-LD context resource for EPCIS/CBV 2.0 includes within scoped contexts mappings such as:

Format	EPCIS 2.0 example for bizTransactionList
	`"bizTransaction": { "@id": "@id", "@type": "@id"}`

10.2.7 Modelling and validating subclasses of EPCIS event

The UML class diagram for EPCIS shows an abstract class EPCISEvent and four subclasses (event types) already defined in v1.2, namely `ObjectEvent`, `AggregationEvent`, `TransactionEvent` and `TransformationEvent`. Version 2.0 of EPCIS introduces a fifth subclass or event type, `AssociationEvent`, which is structurally similar to an `AggregationEvent` but has different semantics; associations are not disassociated when a disaggregation occurs.

In XML, each of these event types or subclasses of `EPCISEvent` has its own designated element, `<ObjectEvent>`, `<AggregationEvent>`, `<TransactionEvent>`, `<TransformationEvent>` and now also `<AssociationEvent>`.

XSD schema bind each of these named elements to a defined type (a reference to a defined structure) through declarations such as:

```
<xsd:element name="ObjectEvent" type="epcis:ObjectEventType"
minOccurs="0" maxOccurs="unbounded"/>
```

XSD also support class inheritance via `xsd:extension` that specifies the base (superclass or abstract event type), e.g.

```
<xsd:complexType name="AggregationEventType">
  <xsd:complexContent>
    <xsd:extension base="epcis:EPCISEventType">
        <xsd:sequence>
          <xsd:element name="parentID" type="epcis:ParentIDType" minOccurs="0"/>
          <xsd:element name="childEPCs" type="epcis:EPCListType"/>
          …
        </xsd:sequence>
        …
  </xsd:complexContent>
</xsd:complexType>
```

JSON has no named elements nor any special syntax to declare that a class or data object is of a specific type.

JSON-LD uses a special keyword, `"@type"` to declare that a class or data object is of a specific type. In JSON-LD, `"@type"` corresponds to the Linked Data predicate `rdf:type`.

The standard context resource for EPCIS/CBV 2.0 defines an alias of the special JSON-LD keyword `"@type"` named `"type"` as part of the effort to hide the JSON-LD special keywords from those users and applications who are primarily interested in processing EPCIS event data as JSON. However, within `bizTransactionList`, `sourceList`, `destinationList` and `sensorReport`, `"type"` SHOULD NOT be mapped to the special JSON-LD keyword `"@type"` that corresponds to the Linked Data predicate `rdf:type`; instead, the standard context resource for EPCIS/CBV 2.0 defines protected scoped contexts for `bizTransactionList`, `sourceList`, `destinationList` and `sensorReport` in which `"type"` is instead mapped via `"@id"` to properties defined within the EPCIS 2.0 ontology, namely `epcis:bizTransactionType` , `epcis:sourceOrDestinationType` (within `sourceList` or `destinationList`) or `epcis:measurementType` (within `sensorReport`).

JSON Schema structures have been defined for the superclass EPCISEvent and for each of the subclasses (event types such as ObjectEvent, AggregationEvent), which reference the validation rules for the superclass EPCISEvent and add their own specific validation rules.

Because JSON has no special way of declaring that a data object is of a specific type, extra care is needed when writing validation rules in JSON Schema to ensure that each set of validating rules will only be tested against the corresponding event type and to indicate that the "type" (rdf:type / @type) field is more important than other fields for validation purposes, in order to select the appropriate validation rules for each EPCIS event type.

By using the if/then/else feature introduced in v7 of JSON Schema, it is possible to use the "if" clause to require a match on the value of "type" and use the "then" clause to reference the validation rule structure defined for that event type.

This approach ensures that each event type is correctly validated without resulting in spurious validation errors for all other EPCIS event types.

Shape Constraint Language (SHACL) is used to validate the JSON-LD representation. SHACL does support inheritance if the data being validated expresses rdfs:subClassOf relationships explicitly. However, in EPCIS 2.0 event data, there are no such declarations *within the EPCIS data* that an epcis:ObjectEvent is a rdfs:subClassOf epcis:Event , nor can this subclass relationship be defined within the JSON-LD context resource.

Instead, the SHACL validation file for EPCIS 2.0 uses sh:targetClass to specify the class to which the validation rules apply, e.g. epcis:AggregationEventShape

```
a sh:NodeShape ;
sh:targetClass epcis:AggregationEvent ;
```

The validation rules from the base class EPCISEvent that are inherited by each subclass event type are simply referenced from within the SHACL shape that is defined for each event type / subclass and such validation rules on core fields of EPCISEvent are simply replicated for each EPCIS event type / subclass.

In SHACL, a node shape (sh:NodeShape) constrains the structure of a class or data object. The SHACL validation file for EPCIS 2.0 contains some node shapes that reference other node shapes. For example, the node shape epcis:SensorElementShape references node shapes epcis:SensorMetadataShape and epcis:SensorReportShape. It does so by using sh:property to reference these two node shapes. Within each referenced node shape (e.g. epcis:SensorMetadataShape and epcis:SensorReportShape), sh:path is used to match the property (e.g. epcis:sensorMetadata or epcis:sensorReport) whose value is constrained by that node shape.

10.2.8 Comparison of how validation rules are expressed in XSD, JSON Schema and SHACL

Validation Rule	XSD	JSON Schema	SHACL
Applies to named field	name="*fieldname*" type="*defined_type*"	Fieldname appears within list of "properties"	sh:path fieldname
Mandatory Field	minOccurs="1" (**may be omitted** since default values of minOccurs="1")	Include fieldname within the list of "required" properties	sh:minCount 1 (**must be asserted** since default value of sh:minCount=0)
Note on default values for minOccurs, maxOccurs and sh:minCount, sh:maxCount	*default of XSD minOccurs = 1, maxOccurs = 1)*	*Only properties specified within "required" are considered mandatory*	*default of sh:minCount = 0, default of sh:maxCount = unbounded*
Optional Field	minOccurs="0" (**must be asserted** since default value of minOccurs="1")	Include fieldname within the list of "properties" but omit from list of "required" properties	sh:minCount 0 (**may be omitted** since default value of sh:minCount=0)

Validation Rule	XSD	JSON Schema	SHACL
Field expects a string	type="xsd:string"	"type":"string"	sh:datatype xsd:string
Field expects a dateTimeStamp	type="xsd:dateTimeStamp"	"type":"string", "format":"date-time"	sh:datatype xsd:dateTimeStamp
Field expects an integer	type="xsd:int"	"type":"integer"	sh:datatype xsd:int
Field expects a decimal value	type="xsd:decimal"	"type":"number"	sh:datatype xsd:decimal
Field expects a double-precision floating-point value	type="xsd:double"	"type":"number"	sh:datatype xsd:double
Field expects a string from a restricted list of enumerated values	references an xsd:simpleType with an xsd:restriction base="xsd:string" containing xsd:enumeration child elements that express the permitted values e.g. <xsd:simpleType name="ActionType"> <xsd:restriction base="xsd:string"> <xsd:enumeration value="ADD"/> <xsd:enumeration value="OBSERVE"/> <xsd:enumeration value="DELETE"/> </xsd:restriction> </xsd:simpleType>	"type":"string", "enum":[…] e.g. "type":"string", "enum":["ADD", "OBSERVE", "DELETE"]	sh:datatype xsd:string sh:in ("ADD" "OBSERVE" "DELETE") e.g. epcis:ActionShape a sh:PropertyShape ; sh:path epcis:action ; sh:name "action" ; sh:datatype xsd:string ; sh:in ("ADD" "OBSERVE" "DELETE") ; sh:minCount 1 ; sh:maxCount 1 ; sh:message "Within ObjectEvent, AggregationEvent, TransactionEvent, AssociationEvent, action is mandatory, and must be a string value, one of either 'ADD', 'OBSERVE' or 'DELETE'" ;
Field expects a URI (in situations where CBV 2.0 does not define a code list)	type="xsd:anyURI" or references an xsd:simpleType with an xsd:restriction base="xsd:anyURI"	"type":"string", "format":"uri"	sh:nodeKind sh:IRI sh:pattern "^(.+?):(.+)$"

Field expects a URI (in situations where CBV 2.0 defines a code list for standard values but custom URI values are also permitted)	type="xsd:anyURI" or references an xsd:simpleType with an xsd:restriction base="xsd:anyURI"	"anyOf": [
		{
		"type":"string",
		"format":"uri"
		},
		{
		"enum": [
		"accepting",
		"arriving",
		…
		"void_shipping"
		]
		}
		]
			sh:nodeKind sh:IRI
			sh:pattern "^(.+?):(.+)$"

10.2.9 Mapping core SBDH fields to the JSON/JSON-LD data format for EPCIS

Section 9.2 explains the optional use of the Standard Business Document Header [SBDH] within the XML representation of an EPCISDocument. A JSON/JSON-LD format of SBDH has not been defined by UN CEFACT or by any other standards organisation. In order to support existing XML users of SBDH within EPCISDocument, three optional properties have been defined for use within the root level of an EPCISDocument in JSON/JSON-LD format. These are shown in the table below.

Optional property within JSON/JSON-LD format at root level within an EPCISDocument	Expected data type	Corresponding XML element in SBDH
epcis:sender	xsd:string	sbdh:Sender
epcis:receiver	xsd:string	sbdh:Receiver
epcis:instanceIdentifier	xsd:string	sbdh:InstanceIdentifier

The mandatory property within EPCISDocument 'creationDate' expresses the date and time of creation of the EPCISDocument as an `xsd:dateTimeStamp` value, equivalent to the SBDH element `sbdh:CreationDateAndTime`.

10.2.10 Online validation tools for JSON Schema and SHACL

Online tools currently available for validation using JSON Schema include:

- https://www.jsonschemavalidator.net/
- https://json-schema-validator.herokuapp.com/
- https://www.liquid-technologies.com/online-json-schema-validator
- https://jsonschemalint.com/

Online tools currently available for validation using SHACL include:

- https://shacl-playground.zazuko.com/
- http://rdfshape.herokuapp.com/

10.2.11 Libraries and toolkits providing JSON-LD support

A number of libraries and toolkits are now available to support JSON-LD in various programming and scripting languages. A list of these is currently provided at https://json-ld.org/#developers.

10.3 Validation schema (references to normative content)

The JSON representation of EPCIS 2.0 data SHALL validate against the JSON Schema, published at https://ref.gs1.org/standards/epcis/2.0.0/epcis-json-schema.json.

The JSON-LD representation of EPCIS 2.0 data SHALL validate against the SHACL file, published at https://ref.gs1.org/standards/epcis/2.0.0/epcis-shacl.ttl.

Section 10.2 provides detailed explanation of the structure of these validation files and how validation rules expressed only previously in XSD are now also expressed within JSON Schema and SHACL.

Note that both the JSON Schema and SHACL validation files take an open shape approach to validation, unlike the closed shape approach taken in XSD validation of the XML data format. This means that additional terms from other namespaces may be used within the JSON/JSON-LD representation of EPCIS 2.0 data and will simply be ignored by the validation files. While XML, JSON and JSON-LD all support the serialisation or expression of hierarchical data structures, XML takes a document-centric approach placing great emphasis on the sequence in which elements appear. JSON and JSON-LD express graph data structures in which the sequence of appearance is insignificant for any two fields / keys at the same level of hierarchy within a JSON object / dictionary.

The **EPCIS 2.0 JSON-LD context file** is published at https://ref.gs1.org/standards/epcis/2.0.0/epcis-context.jsonld.

10.4 Non-normative examples in JSON and JSON-LD

JSON and JSON-LD examples are published at https://ref.gs1.org/docs/epcis/examples/.

11 Bindings for core capture operations module

This section defines bindings for the Core Capture Operations Module. All bindings specified here are based on the XML representation of events defined in section 9.5. An implementation of EPCIS MAY provide support for one or more Core Capture Operations Module bindings as specified below.

11.1 Message queue binding

This section defines a binding of the Core Capture Operations Module to a message queue system, as commonly deployed within large enterprises. A message queue system is defined for the purpose of this section as any system which allows one application to send a message to another application. Message queue systems commonly support both point-to-point message delivery and publish/subscribe message delivery. Message queue systems often include features for guaranteed reliable delivery and other quality-of-service (QoS) guarantees.

Because there is no universally accepted industry standard message queue system, this specification is designed to apply to any such system. Many implementation details, therefore, necessarily fall outside the scope of this specification. Such details include message queue system to use, addressing, protocols, use of QoS or other system-specific parameters, and so on.

An EPCIS implementation MAY provide a message queue binding of the Core Capture Operations Module in the following manner. For the purposes of this binding, a "capture client" is an EPCIS Capture Application that wishes to deliver an EPCIS event through the EPCIS Capture Interface, and a "capture server" is an EPCIS Repository or EPCIS Accessing Application that receives an event from a capture client.

A capture server SHALL provide one or more message queue endpoints through which a capture client may deliver one or more EPCIS events. Each message queue endpoint MAY be a point-to-point queue, a publish/subscribe topic, or some other appropriate addressable channel provided by the message queue system; the specifics are outside the scope of this specification.

A capture client SHALL exercise the `capture` operation defined in section 8.1.2 by delivering a message to the endpoint provided by the capture server. The message SHALL be one of the following:

- an XML document whose root element conforms to the `EPCISDocument` element as defined by the schema of section 9.5; or
- an XML document whose root element conforms to the `EPCISQueryDocument` element as defined by the schema of section 13.1, where the element immediately nested within the `EPCISBody` element is a `QueryResults` element, and where the `resultsBody` element within the `QueryResults` element contains an `EventList` element.

JSON/JSON-LD equivalents can be validated by their respective schema.

An implementation of the capture interface SHALL accept the `EPCISDocument` form and SHOULD accept the `EPCISQueryDocument` form. An implementation of the capture interface SHALL NOT accept documents that are not valid as defined above. Successful acceptance of this message by the server SHALL constitute capture of all EPCIS events included in the message.

Message queue systems vary in their ability to provide positive and negative acknowledgements to message senders. When a positive acknowledgement feature is available from the message queue system, a positive acknowledgement MAY be used to indicate successful capture by the capture server. When a negative acknowledgement feature is available from the message queue system, a negative acknowledgement MAY be used to indicate a failure to complete the capture operation.

Failure may be due to an invalid document, an authorisation failure as described in section 8.1.1, or for some other reason. The specific circumstances under which a positive or negative acknowledgement are indicated is implementation-dependent. All implementations, however, SHALL either accept all events in the message or reject all events.

11.2 HTTP binding

This section defines a binding of the Core Capture Operations Module to HTTP [RFC2616].

An EPCIS implementation MAY provide an HTTP binding of the Core Capture Operations Module in the following manner. For the purposes of this binding, a "capture client" is an EPCIS Capture Application that wishes to deliver an EPCIS event through the EPCIS Capture Interface, and a "capture server" is an EPCIS Repository or EPCIS Accessing Application that receives an event from a capture client.

A capture server SHALL provide an HTTP URL through which a capture client may deliver one or more EPCIS events.

A capture client SHALL exercise the `capture` operation defined in section 8.1.2 by invoking an HTTP POST operation on the URL provided by the capture server. The message payload SHALL be one of the following:

- an XML document whose root element conforms to the `EPCISDocument` element as defined by the schema of section 9.5; or
- an XML document whose root element conforms to the `EPCISQueryDocument` element as defined by the schema of section 13.1, where the element immediately nested within the `EPCISBody` element is a `QueryResults` element, and where the `resultsBody` element within the `QueryResults` element contains an `EventList` element.

An implementation of the capture interface SHALL accept the `EPCISDocument` form and SHOULD accept the `EPCISQueryDocument` form. An implementation of the capture interface SHALL NOT accept documents that are not valid as defined above. Successful acceptance of this message by the server SHALL constitute capture of all EPCIS events included in the message.

Status codes returned by the capture server SHALL conform to [RFC2616], section 10. In particular, the capture server SHALL return status code 200 to indicate successful completion of the capture operation, and any status code 3xx, 4xx, or 5xx SHALL indicate that the capture operation was not successfully completed. The specific circumstances under which a success or failure code is returned are implementation-dependent. All implementations, however, SHALL either accept all events in the message or reject all events.

12 REST Bindings

12.1 Code conventions

Complete API specification: The complete specification corresponding to the REST Bindings using the OpenAPI syntax is published at https://ref.gs1.org/standards/epcis/2.0.0/openapi.json

Media types: To avoid overloading examples with repetitive content descriptions, examples only use the media type `application/json` instead of the complete list `application/json`, `application/ld+json` and `application/xml`.

12.2 Introduction to REST

EPCIS 1.x was built on SOAP, a stateful XML-based messaging protocol for distributed enterprise environments.

EPCIS 2.0 adds a RESTful protocol, optimising the web-based capture and query of EPCIS events. With EPCIS 2.0, the Web becomes a platform for a frictionless integration of business processes in complex supply chain information systems, as well as consumer-facing applications.

The EPCIS REST bindings provide an interface based on Representational State Transfer (REST) architecture style [REST]. REST is the architectural principle that lies at the heart of the Web. It shares a similar goal with SOAP [SOAP] interfaces already defined in the EPCIS standard, which is to increase interoperability for a looser coupling between the parts of distributed applications. However, the goal of REST is to achieve this in a more lightweight and simpler manner, focussing on resources rather than on functions (as is the case with SOAP Web services). In particular, REST uses the Web as an application platform and fully leverages all the features inherent to HTTP such as authentication, authorisation, encryption, compression, and caching. This way, REST brings services to the browser and modern Web languages: resources can be queried, linked and bookmarked and the results are visible with any Web browser or Web tool with no need to generate complex source code out of WSDL files to be able to interact with the service. The EPCIS RESTful interface uses the stateless HTTP [HTTP] protocol. In REST, URIs not only address individual resources, such as Web pages, but also collections that can be accessed via the Web using a small number of simple HTTP verbs such as `GET`, `POST`, `PUT`, `DELETE`. This means REST URIs in EPCIS 2.0 refer to individual EPCIS events, EPCIS event types and collections of EPCIS events.

This interface is an alternative to the SOAP query control interface already defined in the EPCIS standard. REST complements the SOAP interface and is meant to foster use cases where modern Web technologies (e.g., Javascript clients or mobile applications) are used to interact with an EPCIS repository, with the ability to retrieve EPCIS event data via simple Web requests. A new endpoint exposes EPCIS events types and individual events as REST resources. Furthermore, queries can also be treated as resources in their own right, each described by its URL [URL].

To summarise, the EPCIS 2.0 REST bindings offer:

- Bulk capturing of events through the /`capture` interface
- Complex queries with the possibility to define query trigger rules (i.e., for a streaming query) and subscriptions
- Events endpoints
- Top-level resources endpoints

all through a RESTful interface.

Table 12-1 Overview of EPCIS 2.0 endpoints

Endpoint	OPTIONS	GET	POST	DELETE	Equivalent `SimpleEventQuery` parameter

					(section 8.2.7), where applicable
/	X				
/capture	X	X	X		
/capture/{captureID}	X	X			
/events	X	X	X		
/events/{eventID}	X	X			
/eventTypes	X	X			
/eventTypes/{eventType}	X	X			eventType
/eventTypes/{eventType}/events	X	X			
/epcs	X	X			
/epcs/{epc}	X	X			
/epcs/{epc}/events	X	X			
/bizSteps	X	X			
/bizSteps/{bizStep}	X	X			
/bizSteps/{bizStep}/events	X	X			EQ_bizStep
/bizLocations	X	X			
/bizLocations/{bizLocation}	X	X			
/bizLocations/{bizLocation}/events	X	X			EQ_bizLocation
/readPoints	X	X			
/readPoints/{readPoint}	X	X			
/readPoints/{readPoint}/events	X	X			EQ_readPoint
/dispositions	X	X			
/dispositions/{disposition}	X	X			
/dispositions/{disposition}/events	X	X			EQ_disposition
/queries	X	X	X		
/queries/{queryName}	X	X		X	
/queries/{queryName}/subscriptions	X	X	X		
/queries/{queryName}/subscriptions/{subscriptionID}	X	X		X	
/queries/{queryName}/events	X	X			
/nextPageToken/{token}				X	

Each endpoint, shown in the table above, will be described in a machine-readable way using the Open API 3 [OpenAPI] API description format. The OpenAPI interface description for EPCIS will be a formal artefact of the EPCIS 2.0 standard and linked from the EPCIS landing page at https://www.gs1.org/epcis. It plays a similar logical role to the WSDL interface described in section 13.

Note that `/events` is the simplest REST endpoint through which a `SimpleEventQuery` can be performed.

The RESTful API for EPCIS has been designed for easy discoverability of related endpoints, by aligning with the principles of Hypertext As The Engine Of Application State [HATEOAS]. For example, the following are examples of endpoints that do not return data – they simply point to one or more related endpoints (that further extend the URI path information) that do provide data, such as collections of events:

- /eventTypes/{eventType} points to /eventTypes/{eventType}/events
- /eventTypes/{eventType} points to /eventTypes/{eventType}/events
- /bizSteps/{bizStep} points to /bizSteps/{bizStep}/events

So just as a human can read a Web page and follow hyperlinks to related resources, software interacting with a RESTful API that supports HATEOAS can start at any endpoint and may be able to iteratively discover additional endpoints that may be of interest.

12.3 Content negotiation, service discovery and custom headers for EPCIS

Each endpoint SHALL support HTTP content negotiation [HTTPSemanticsContent] for at least JSON [JSON] and JSON-LD [JSONLD], MAY support XML [XML] and MAY support HTML [HTML] and any other formats. This is done by setting the value of the `Accept` header, e.g., to `application/json`, `application/ld+json`, `application/xml` or `text/html` [RFC7231, RFC7303]. All responses SHALL return a full EPCIS Document containing the list of events within the EPCIS Body. If the client requests a media type that the server does not support, the server SHALL reply with HTTP status code `406 Not Acceptable`.

EPCIS 2.0 specifies custom headers for client and server to agree on the EPCIS version, vendor version and EPCIS and CBV extensions and the version of related resources. For all endpoints, an EPCIS server SHALL support `GS1-CBV-Version`, `GS1-EPC-Format`, `GS1-CBV-Format`, `GS1-EPCIS-Version`, `GS1-EPCIS-Min`, `GS1-EPCIS-Max` and `GS1-Extensions`. By convention, if a client omits the EPCIS version or EPCIS version min/max range, the server SHALL use the EPCIS version defined by the `GS1-EPCIS-Version` header. A server MAY support `GS1-Extensions` and `GS1-Vendor-Version`. For the `/capture` endpoint, the server SHALL additionally support `GS1-EPCIS-Capture-Limit`, to specify the maximum number of events that can be captured per call and SHALL support `GS1-EPCIS-Capture-File-Size-Limit` to specify the EPCIS document size in bytes / octets. A server SHALL support `GS1-Capture-Error-Behaviour` to declare the error behaviour of the capture interface. The default value of `GS1-Capture-Error-Behaviour` SHALL be `rollback`. Each custom header is described in the table below.

Table 12-2 EPCIS and CBV headers

Header	Description	Examples	Request	Response
GS1-EPCIS-Version	The EPCIS version	1.0.0, 1.1.0, 1.2.0, 2.0.0	X	X
GS1-EPCIS-Min	The lowest EPCIS version supported	>=1.0.0	X	X
GS1-EPCIS-Max	The highest EPCIS version supported	1.0.0, 1.1.0, 1.2.0, 2.0.0	X	X
GS1-CBV-Version	The core business vocabulary version	1.2.2, 2.0.0	X	X
GS1-EPC-Format	Request or response header to indicate whether EPCs are expressed as GS1 Digital Link URIs or as EPC URNs.	Always_DL	X	X
GS1-CBV-XML-Format	Request or response header to indicate whether CBV URI fields are expressed in XML as URLs or URNs.	Always_URL	X	X
GS1-Extensions	Specific extensions supported (e.g. for FIT) such as CBV or EPCIS extensions.	"example-epc-ext":"http://example.com/epcis/"	X	X
GS1-Vendor-Version	A versioning scheme that can be freely chosen by the vendor	example-version-1.0	X	X
GS1-EPCIS-Capture-Limit	The maximum number of EPCIS events that can be captured per call	500		X
GS1-EPCIS-Capture-File-Size-Limit	The maximum event document length in octets (8-bit bytes)	1024		X

Header	Description	Examples	Request	Response
GS1-Query-Min-Record-Time	An optional header to specify the smallest possible record time for EPCIS events in a query subscription.	2020-04-04T20:33:31.116-06:00		X
GS1-Capture-Error-Behaviour	A header to control how the capture interface will behave in case of an error: ▪ rollback: "All or nothing". Either the capture job is entirely successful or all EPCIS events are rejected. ▪ proceed: "Greedy capture". The capture interface tries to capture as many EPCIS events as possible, even if there are errors. The default behaviour is rollback, as in EPCIS 1.2.	rollback	X	X
GS1-Next-Page-Token-Expires	The expiry time for nextPageToken. This header is optional.	2021-12-08T14:58:56.591Z		X
GS1-Signature	The signature for event pushed via Webhook subscriptions is contained in the **GS1-Signature** HTTP header of the server request.			

Each endpoint SHALL respond to the OPTIONS verb by returning the list of allowed HTTP verbs as well as supported headers for a resource and their default values. This feature was added to provide service discovery and make EPCIS 2.0 future proof by supporting granular versions of EPCIS, CBV and extensions.