The Global Language of Business

EPCIS Standard

enables disparate applications to create and share visibility event data, both within and across enterprises.

Release 2.0, COMMUNITY REVIEW DRAFT, October 2021

Document Summary

Document Item	Current Value
Document Name	EPCIS Standard
Document Date	October 2021
Document Version	2.0
Document Issue	
Document Status	COMMUNITY REVIEW DRAFT
Document Description	enables disparate applications to create and share visibility event data, both within and across enterprises.

4 Contributors

Name	Company
Christophe Devins	Adents
Roula Karam	Antares Vision
Harald Sundmaeker	ATB Institut für angewandte Systemtechnik Bremen GmbH
Junyu Wang	Auto-ID Labs at Fudan University
Jaewook Byun	Auto-ID Labs at KAIST
Sangtae Kim	Auto-ID Labs at KAIST
Yalew Tolcha	Auto-ID Labs at KAIST
Jeanne Duckett	Avery Dennison RFID
Cyrille Bordier	Axway
Marcus Chang	Axway
Onur Önder	BLG CONTRACT LOGISTICS GmbH & Co. KG
Laura Weingarten	BLG CONTRACT LOGISTICS GmbH & Co. KG
Trond Saure	Bouvet Norge AS
Philip Allgaier	bpcompass GmbH
Steffen Butschbacher	bpcompass GmbH
Maheshwar Lingichetty	Bracket Global
Hans Peter Scheidt	C & A SCS
Arnaud Kreweras	Carrefour
Robert Celeste	Center for Supply Chain Studies

Name	Company
Jochen Metschke	Ceratizit
Doug Migliori	ControlBEAM Digital Automation / ADC Technologies Group
Ludovic Fargeas	Courbon
Martijn Veerman	Customer Value
David Harper	Delivr Corporation
Mario Mira	Dentsu Aegis Network
Michiel Valee	Dockflow
Philip Heggelund	DuckScape Inc
Jim Springer	EM Microelectronic
Jayson Berryhill	Envisible LLC
Nicolas Becker	European EPC Competence Center GmbH (EECC)
Falk Nieder	European EPC Competence Center GmbH (EECC)
Sebastian Schmittner	European EPC Competence Center GmbH (EECC)
Georg Schwering	European EPC Competence Center GmbH (EECC)
Dominique Guinard	EVRYTHNG
Joël Vogt	EVRYTHNG
Beth Davis	FoodLogiQ
Julie McGill	FoodLogiQ
Kevin Capatch	Geisinger Health System (GHS)
Chris Roberts	GlaxoSmithKline

Name	Company
Gianluca Fazio	GS1 Argentina
Catherine Koetz	GS1 Australia
Bonnie Ryan	GS1 Australia
Sue Schmid	GS1 Australia
Marcel Sieira	GS1 Australia
Stephan Wijnker	GS1 Australia
Eugen Sehorz	GS1 Austria
Luiz Costa	GS1 Brasil
Flavia Costa	GS1 Brasil
Kevin Dean	GS1 Canada
Nicole Golestani	GS1 Canada
Zubair Nazir	GS1 Canada
Connie Wong	GS1 Canada
Yi Ding	GS1 China
jia jianhua	GS1 China
Yan Luo	GS1 China
Yi Wang	GS1 China
zhang wm	GS1 China
XinMin Wu	GS1 China
Ruoyun Yan	GS1 China

Name	Company
James Perng	GS1 Chinese Taipei
James Perng	GS1 Chinese Taipei
Luis Paniagua	GS1 Costa Rica
Jesper Kervin Franke	GS1 Denmark
Petri Leppänen	GS1 Finland
Adrien Molines	GS1 France
Nicolas Pauvre	GS1 France
Mayra Castellanos	GS1 Germany
David Hintzen	GS1 Germany
Sandra Hohenecker	GS1 Germany
Ralph Troeger	GS1 Germany
Roman Winter	GS1 Germany
Henri Barthel	GS1 Global Office
Rosalie Clemens	GS1 Global Office
Nadi (Scott) Gray	GS1 Global Office
Eileen Harpell	GS1 Global Office
Nora Kaci	GS1 Global Office
Timothy Marsh	GS1 Global Office
Gena Morgan	GS1 Global Office
Neil Piper	GS1 Global Office

Name	Company
Craig Alan Repec	GS1 Global Office
Greg Rowe	GS1 Global Office
John Ryu	GS1 Global Office
Kevin Stark	GS1 Global Office
Claude Tetelin	GS1 Global Office
Elena Tomanovich	GS1 Global Office
Jaco Voorspuij	GS1 Global Office
Grace Yang	GS1 Global Office
Benedict Chan	GS1 Hong Kong, China
Chris Lai	GS1 Hong Kong, China
Wai Shing Lai	GS1 Hong Kong, China
wayne Luk	GS1 Hong Kong, China
Zsolt Bocsi	GS1 Hungary
Krisztina Vatai	GS1 Hungary
Mahdi Barati	GS1 Iran
Tim Daly	GS1 Ireland
Alec Tubridy	GS1 Ireland
Emanuela Casalini	GS1 Italy
Giada Necci	GS1 Italy
Linda Vezzani	GS1 Italy

Name	Company
Koji Asano	GS1 Japan
Yoshihiko Iwasaki	GS1 Japan
Kazuna Kimura	GS1 Japan
Noriyuki Mama	GS1 Japan
Yuki Sato	GS1 Japan
Rocio Rivera	GS1 Mexico
Ben Ensink	GS1 Netherlands
Sarina Pielaat	GS1 Netherlands
Reinier Prenger	GS1 Netherlands
Gabriel Sobrino	GS1 Netherlands
Jan Westerkamp	GS1 Netherlands
Gary Hartley	GS1 New Zealand
Klaudiusz Borowiak	GS1 Poland
Zbigniew Rusinek	GS1 Poland
Pedro Lima	GS1 Portugal
Alexey Krotkov	GS1 Russia
Olga Soboleva	GS1 Russia
Alvin Goh	GS1 Singapore
Ferran Domenech Fuste	GS1 Spain
Mats Bjorkqvist	GS1 Sweden

Name	Company
Fredrik Holmström	GS1 Sweden
Heinz Graf	GS1 Switzerland
Raphael Pfarrer	GS1 Switzerland
Shawn Chen	GS1 Thailand
Rami Habbal	GS1 UAE
Sophie Fuller	GS1 UK
Jason Hale	GS1 UK
Neil Aeschliman	GS1 US
Hussam El-Leithy	GS1 US
James Lynch	GS1 US
Melanie Nuce	GS1 US
Vivian Underwood	GS1 US
Alice Nguyen	GS1 Vietnam
Natham Barry	IBM (US)
Gokul Kandiraju	IBM (US)
Roman Vaculin	IBM (US)
Sylvia Rubio Alegren	ICA Sverige AB
Megan Brewster	Impinj, Inc
Patrick Chanez	INEXTO SA
Thomas Burke	Institute of Food Technologists

Name	Company
Sebastian Bartkowiak	Institute of Logistics and Warehousing
Masatoshi Nomachi	Japan Pallet Rental Corporation
Scott Pugh	Jennason LLC
Rosemary Hampton	Johnson & Johnson
Rajendra Kulkarni	Johnson & Johnson
April Anne Sese	Johnson & Johnson
Sean Lockhead	Lockhead Consulting Group LLC
Ted Osinski	MET Laboratories
Alexander Hille	Migros-Genossenschafts-Bund
Marc Inderbitzin	Migros-Genossenschafts-Bund
Mark Harrison	Milecastle Media Limited
Upender Solanki	Movilitas Consulting AG
Oliver Erlenkämper	Movilizer GmbH
Endre Lazar	Movilizer GmbH
Mario Mira	Movilizer GmbH
Danny Haak	Nedap
Vera Feuerstein	Nestlé
Jason Geyen	Optel Group
Michael Natale	Pfizer
Richard Graves	Phy

Name	Company
Matt Glassman	rfXcel Corporation
Graham Stanley	rfXcel Corporation
Jürgen Engelhardt	Robert Bosch GmbH
John Fredrik Reimers	Robert Bosch GmbH
Nikolaos Servos	Robert Bosch GmbH
Jan Reichert	SAP SE
Thomas Rumbach	SAP SE
Marc Damhösl	Schweizerische Bundesbahnen SBB
Dominik Halbeisen	Schweizerische Bundesbahnen SBB
Rémy Höhener	Schweizerische Bundesbahnen SBB
Holger Strietholt	Schweizerische Bundesbahnen SBB
John Walker	Semaku
Vladimir Alexiev	Sirma AI (Ontotext)
Vladimir Dzalbo	Smartrac Technology Germany GmbH
Shreenidhi Bharadwajj	Syndigo
Tony Zhang	Syndigo
Joseph Lipari	Systech International
Octavio Rodriguez	Systech International
Maik Bollmacher	T-Systems International GmbH
Martin Herold	T-Systems International GmbH

Name	Company
Tobias Michelchen	T-Systems International GmbH
Carsten Baumhögger	tabya GmbH
Tobias Müller	tabya GmbH
Gergely Köves	TE-FOOD International GmbH
Eric Moore	Testo
Melissa Banning	TraceLink
Elizabeth Waldorf	TraceLink
Evan Fernando	Tyson
Jay Crowley	US Data Management, LLC (USDM)
Bharat Reddy Vaka	Vaka Consulting Inc
Rob Magee	Vantage Consulting Group
Jussi Numminen	wirepas

5

6

7 Log of Changes

Release	Date of Change	Changed By	Summary of Change
1.0			Initial version
1.1	May 2014		EPCIS 1.1 is fully backward compatible with EPCIS 1.0.1. EPCIS 1.1 includes these new or enhanced features: Support for class-level identification is added to `ObjectEvent`, `AggregationEvent`, and `TransformationEvent` through the addition of quantity lists. A new event type, `TransformationEvent`, provides for the description of events in which inputs are consumed and outputs are produced. The "why" dimension of all event types are enhanced so that information about the sources and destinations of business transfers may be included. The "why" dimension of certain event types are enhanced so that item/lot master data may be included. The `SimpleEventQuery` is enhanced to encompass the above changes to event types. The introductory material is revised to align with the GS1 System Architecture. The XML extension mechanism is explained more fully. The `QuantityEvent` is deprecated, as its functionality is fully subsumed by `ObjectEvent` with the addition of quantity lists.
1.2	Sep 2016		EPCIS 1.2 is fully backward compatible with EPCIS 1.1 and 1.0.1. EPCIS 1.2 includes these new or enhanced features: A mechanism is introduced to declare that a prior EPCIS event is in error, for use when it is impossible to correct the historical trace by means of ordinary EPCIS events. This mechanism includes the `errorDeclaration` structure in an EPCIS event and associated query parameters. An optional `eventID` is added to all EPCIS events. Its main intended use is to allow for an error declaration event to (optionally) refer to one or more corrective events. The Simple Event Query is enhanced to clarify that queries for extension or ILMD fields apply only to top-level XML elements, and a new set of query parameters is introduced to query for XML elements nested within top-level elements. The role of an EPCIS document as a means to transmit events point-to-point is clarified. The EPCIS Header in the XML schemas is enhanced to allow for optional inclusion of master data. The use of extension elements within `<readPoint>` and `<bizLocation>` is deprecated. Section 14 regarding conformance is added.

Release	Date of Change	Changed By	Summary of Change
2.0	October 2020		Major release EPCIS 2.0 in conjunction with CBV 2.0, including: • Addition of JSON/SON-LD syntax (alongside XML) • Addition of REST bindings (alongside SOAP/WSDL) • Completely overhauled UML diagram • Clarification on distinction between standard vocabulary and user vocabulary • New AssociationEvent • New "How" event dimension • Overview of EPCIS event "dimensions" with cross references to relevant sections in EPCIS & CBV • New Persistent Disposition indicates non-transient business state of an object • New SensorElement to accomodate sensor data • Additon of certificationInfo to core EPCISEvent • Updated SimpleEventQuery parameters • Removal of support for Simple Master Data Query and EPCIS Master Data Document

8

9

Disclaimer

GS1®, under its IP Policy, seeks to avoid uncertainty regarding intellectual property claims by requiring the participants in the Work Group that developed this **EPC Information Services (EPCIS) Standard** to agree to grant to GS1 members a royalty-free licence or a RAND licence to Necessary Claims, as that term is defined in the GS1 IP Policy. Furthermore, attention is drawn to the possibility that an implementation of one or more features of this Specification may be the subject of a patent or other intellectual property right that does not involve a Necessary Claim. Any such patent or other intellectual property right is not subject to the licencing obligations of GS1. Moreover, the agreement to grant licences provided under the GS1 IP Policy does not include IP rights and any claims of third parties who were not participants in the Work Group.

Accordingly, GS1 recommends that any organisation developing an implementation designed to be in conformance with this Specification should determine whether there are any patents that may encompass a specific implementation that the organisation is developing in compliance with the Specification and whether a licence under a patent or other intellectual property right is needed. Such a determination of a need for licencing should be made in view of the details of the specific system designed by the organisation in consultation with their own patent counsel.

Table of Contents

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 Comprehensive Business Vocabulary (CBV) standard [CBV2.0]. 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.

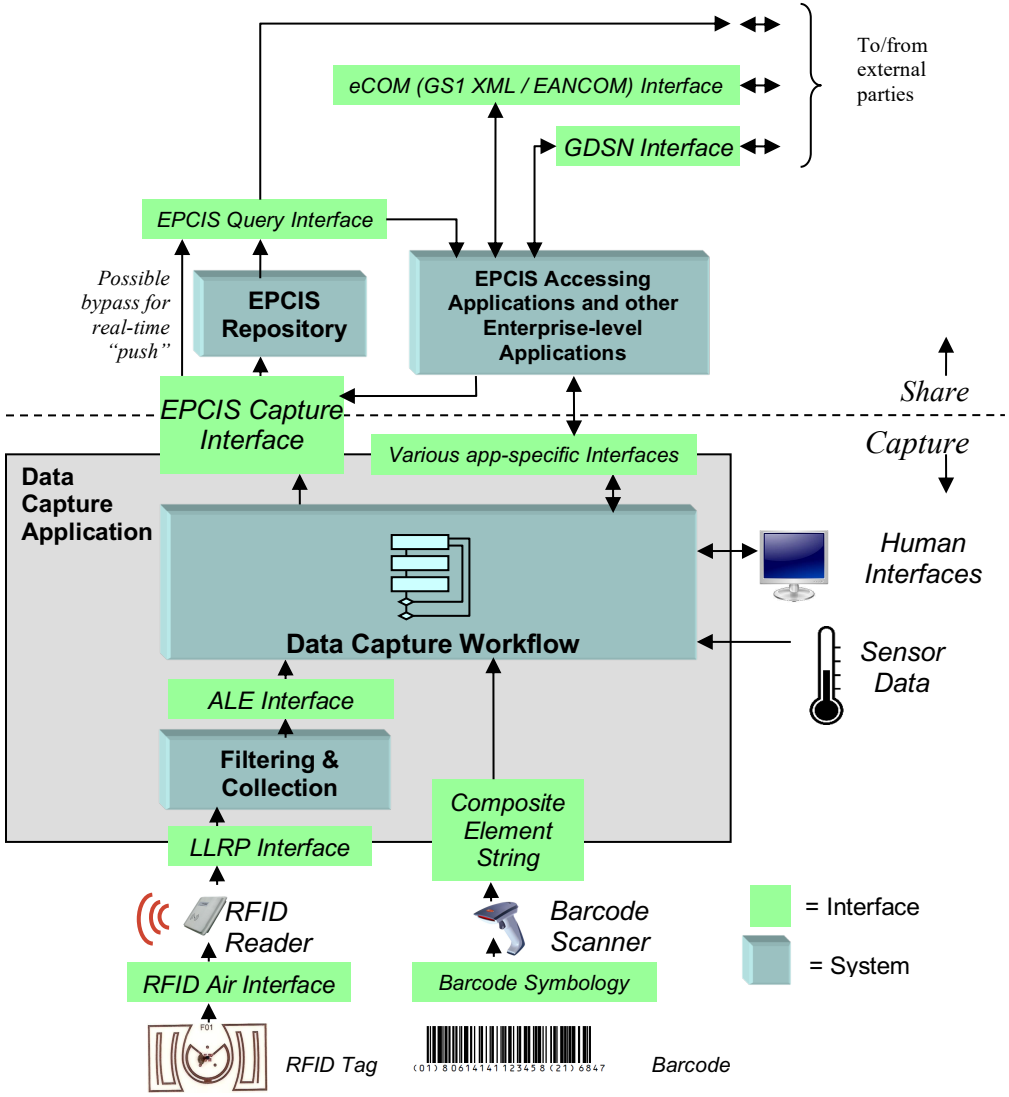
GS1 standards may be divided into the following groups according to their role in supporting information needs related to real-world entities in supply chain business processes:

- 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

The diagram to the right 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

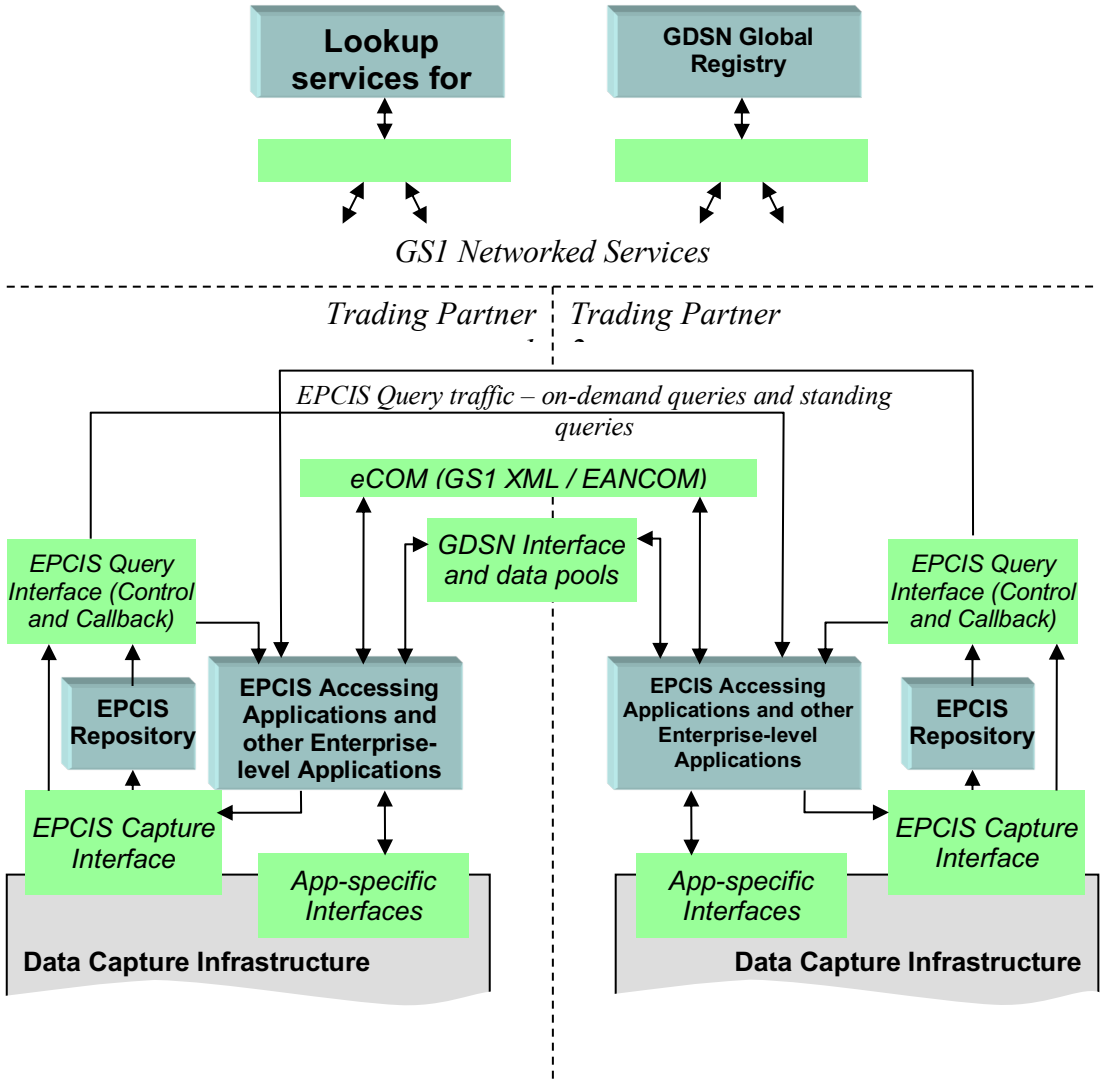
GS1 standards in the "Share" layer pertain to three categories of data that are shared between end users:

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.

The figure to the right 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 § 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-enabled 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-enabled 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 Comprehensive 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, NEED NOT, CAN, and CANNOT are to be interpreted as specified in § 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.

➢ Placeholders for changes that need to be made to this document prior to its reaching the final stage of approved GS1 standard are prefixed by a rightward-facing arrowhead, as this paragraph is.

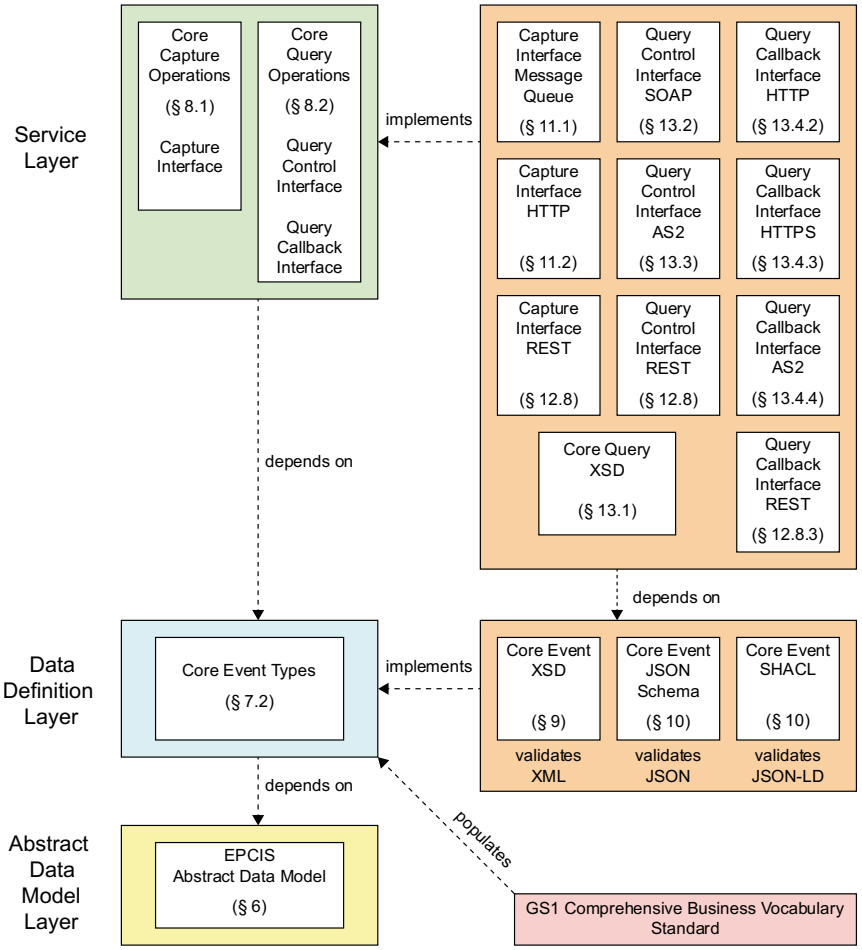
Links to gs1 online resources whose URL must still be verified prior to publication of this document are highlighted in yellow.

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 to the right and described below.



- *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 Comprehensive Business Vocabulary Standard:* The GS1 Comprehensive 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.

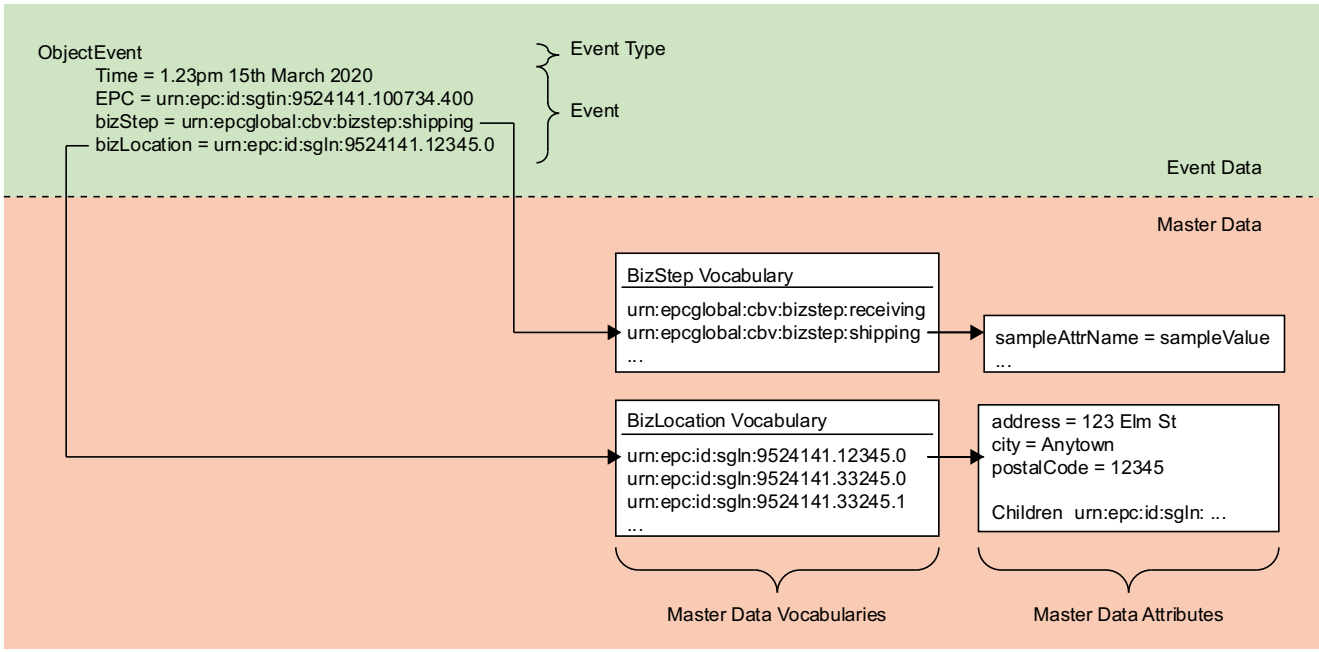
6.1 Event data and master data

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.

The structure of event data and master data in EPCIS is illustrated to the right. (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 § 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 (§ *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:

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 § *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

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 § *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 § 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:

How extension is dsseminated	Responsible organisation	Extension method				
		New Event Type	New Event Field	New Vocabulary Type	New master data or ILMD (§ 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)

How extension is dsseminated	Responsible organisation	Extension method				
		New Event Type	New Event Field	New Vocabulary Type	New master data or ILMD (§ 7.3.8) Attribute	New Vocabulary Element
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 [RFC2396]. 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* [GS1DL1.1], 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 § 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. That is, the "direct or indirect descendants" of a vocabulary element are the set of vocabulary elements obtained by taking the transitive closure of the "children" relation starting with the given vocabulary element.

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 (§ *8.2*), and may play a role in carrying out authorisation policies (§ *8.2.2*), but do not otherwise add any additional complexity or mechanism to the Abstract Data Model Layer.

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 § *0* 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 § *0*:

- *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 § *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 § *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 § *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

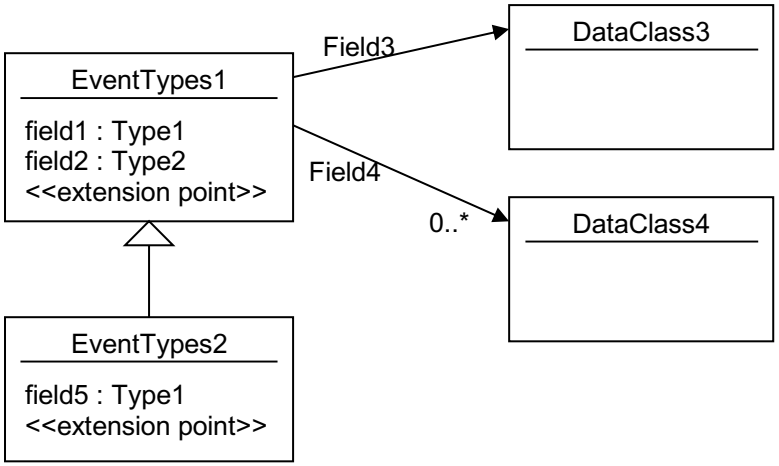
extend existing Vocabulary Types, and Vocabulary Elements that populate existing Vocabularies. Other groups may also occasionally define Vocabulary Types.

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 Types 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.

The example UML diagram to the right shows a data definition for two Event Types, `EventTypes1` and `EventTypes2`. 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 `EventTypes1` in this example has four fields. `Field1` and `Field2` are of primitive type `Type1` and `Type2` respectively. `EventTypes1` has another field `Field3` whose type is `DataClass3`. Finally, `EventTypes1` 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 `EventTypes2`. The arrow with the open-triangle arrowhead indicates that `EventTypes2` is a subclass of `EventTypes1`. This means that `EventTypes2` actually has five fields: four fields inherited from `EventTypes1` 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 § 9.1.

All definitions of Event Types SHALL include an extension point, to provide for the extensibility defined in § 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 § 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 § 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 § 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 § 7.1.1, are as follows:

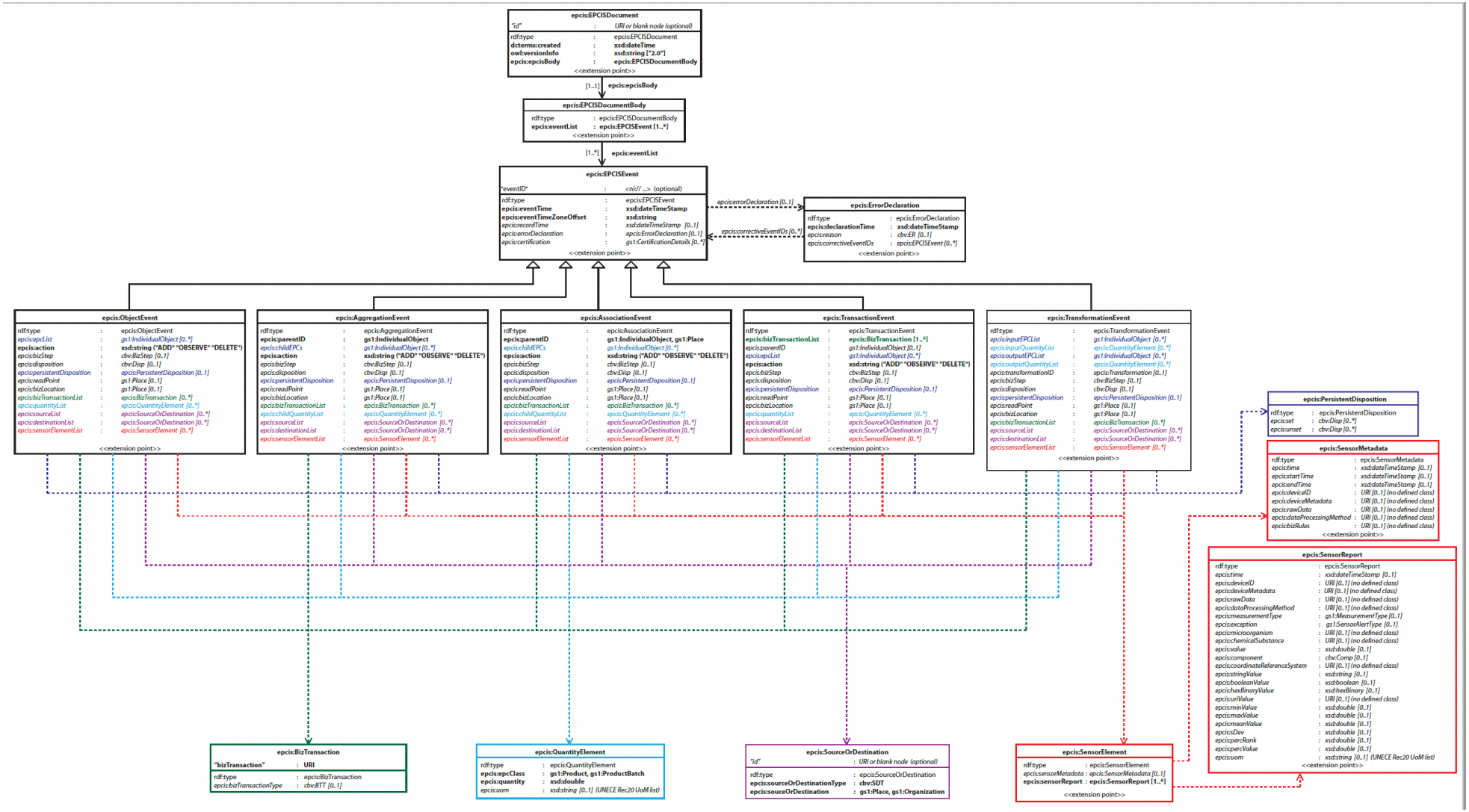
- *Value Types:* Primitive types defined in § 7.3.1 and 7.3.2.
- *Event Types:* Event types as shown in the UML diagram below, and defined in § 7.4.1 through 0.
- *Event Fields:* Included as part of the Event Types definitions.
- *Vocabulary Types:* Types defined in § 7.3.3 through 7.3.6, and summarised in Section 7.2.
- *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 (§ 7.3.6), the `Disposition` vocabulary (§ 7.3.6.2), and the `BusinessTransactionType` vocabulary (§ 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` (§ 7.4.1) is a generic base class for all event types in this module as well as others.
- `ObjectEvent` (§ 7.4.1.2) represents an event that happened to one or more physical or digital objects.
- `AggregationEvent` (§ 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` (§ 7.4.4) represents an event in which one or more objects become associated or disassociated with one or more identified business transactions.
- `TransformationEvent` (§ 0) 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` (§ 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 Diagram of EPCIS Event Types



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; (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 § *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.

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

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 7.4.5 TransactionEvent	*EPC Tag Data Standard (TDS) section 6,"EPC URI"*
			parentID	7.4.3 AggregationEvent 7.4.4 TransactionEvent 7.4.6 AssociationEvent	8.2 Physical or Digital Objects (Instance)
			childEPCs	7.4.3 AggregationEvent 7.4.6 AssociationEvent	
			inputEPCList	7.4.5 TransformationEvent	
			outputEPCList		
		Class	quantityList	7.4.2 ObjectEvent	*EPC Tag Data Standard (TDS) § 8,"URIs for EPC Pure Identity Patterns"*
			childQuantityList	7.4.3 AggregationEvent 7.4.6 AssociationEvent	
			inputQuantityList	7.4.5 TransformationEvent	8.3 Physical or Digital Objects (Class)
			outputQuantityList		

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 TransactionEvent 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 TransactionEvent 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	7.1 Business Steps
		bizTransactionList		8.5 Business Transactions

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
		disposition	7.4.5 TransactionEvent 7.4.6 AssociationEvent	7.1 Dispositions
		persistentDisposition		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	
(XformationID)		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

788

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.

7.2.4 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	7.3.4	User	urn:epcglobal:epcis:vtype:ReadPoint
BusinessLocationID	7.3.4	User	urn:epcglobal:epcis:vtype:BusinessLocation
BusinessStepID	7.3.6	Standard	urn:epcglobal:epcis:vtype:BusinessStep
DispositionID	7.3.6.2	Standard	urn:epcglobal:epcis:vtype:Disposition
BusinessTransaction	7.3.6.3.2	User	urn:epcglobal:epcis:vtype:BusinessTransaction
BusinessTrasactionTypeID	7.3.6.3.1	Standard	urn:epcglobal:epcis:vtype:BusinessTransactionType
EPCClass	7.3.6.4	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	7.3.4	User	urn:epcglobal:epcis:vtype:Location
PartyID	7.3.4	User	urn:epcglobal:epcis:vtype:Party
ErrorReasonID	7.4.1.2	Standard	urn:epcglobal:epcis:vtype:ErrorReason
SensorPropertyTypeID	7.3.7.1.3	Standard	urn:epcglobal:epcis:vtype:SensorPropertyType
MicroorganismID	7.3.7.1.5	User	urn:epcglobal:epcis:vtype:Microorganism
ChemicalSubstanceID	7.3.7.1.6	User	urn:epcglobal:epcis:vtype:ChemicalSubstance
ResourceID	7.3.7.1.7	User	urn:epcglobal:epcis:vtype:Resoruce

7.3 Core event types module – building blocks

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

7.3.1 Primitive types

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

Type	Description
`int`	An integer. Range restrictions are noted where applicable.
`Time`	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 compliant representation SHOULD be used.
`EPC`	An Electronic Product Code, as defined in [TDS]. Unless otherwise noted, EPCs are represented in "pure identity" URI form as defined in [TDS], .

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 (§ *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 § *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` (§ *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

§ 8 of the CBV defines the data fields and the expected value types and data types used in the "What" dimension of EPCIS events.

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`	`Decimal`	(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 § *0*. Any EPC whose structure incorporates the concept of object class can be referenced as an EPCClass. The standards for SGTIN EPCs are elaborated below.

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 § 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 Comprehensive 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 specifed in § 7.4.1.

7.3.4.1 The "When" dimension in the Error Declaration

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

7.3.4.2 The "When" dimension in Sensor Metadata

EPCIS Sensor Metadata, specified in § 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), as summarised here:

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 overtly 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 § *0*. 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`, `BusinessStepID`, 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.

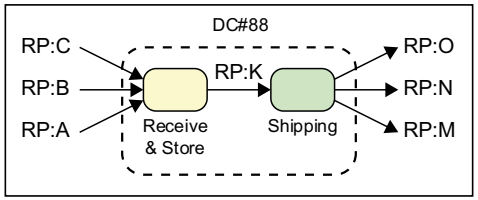
Non-Normative: Explanation: Logical Reader and Physical Reader are omitted from the definitions of EPCIS events in this specification. Physical Reader is generally not useful information for exchange between partners. For example, if a reader malfunctions and is replaced by another reader of identical 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.

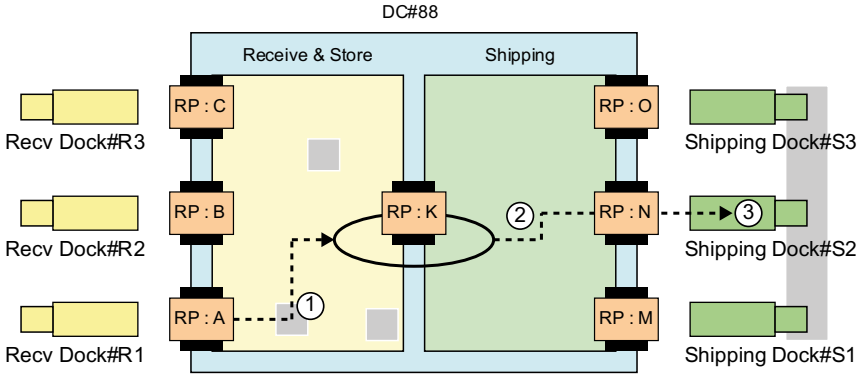
901

902 **7.3.5.1 Example of the distinction between a read point and a business location (Non-Normative)**

Graph View:



Physical View:



903

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

904

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 § *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 § *0*.

Non-Normative: Explanation: Using an extensible vocabulary for business step identifiers allows GS1 standards (including and especially the GS1 Comprehensive 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.

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 explicity `unset`. Like disposition, persistentDisposition leverages the `DispositionID` Standard Vocabulary.

Non-Normative: Explanation: Using an extensible vocabulary for disposition identifiers allows GS1 standards (including and especially the GS1 Comprehensive 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.

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 § 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 § 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, and furthermore the PostHaste Shipping Company uses 12-digit tracking IDs, then the following business transaction IDs might be associated with a particular EPC over time:

http://transaction.acme.com/po/12345678
http://transaction.acme.com/shipment/34ABC8
urn:posthaste:tracking:123456789012

(In this example, it is assumed that PostHaste Shipping has registered the URN namespace "posthaste" with IANA.) An EPCIS Accessing Application might query EPCIS and discover all three 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 `BusinssTransaction` 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 § 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 § 6.2.

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 § 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).

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 advisible 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 § 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.

Field	Type	Description
`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`.
`value`	`Float`	(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`.

Field	Type	Description
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 § 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.

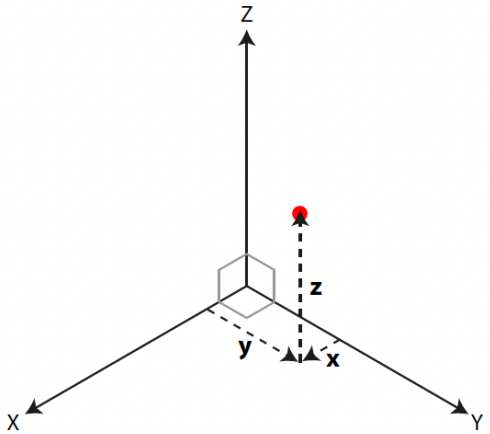
Field	Type	Description
booleanValue	Boolean	(Optional) The Boolean value of the property specified by the type and/or exception field as part of the sensorReport element. If a time field is present, the Boolean value SHALL pertain to this point in time. Otherwise, it SHALL refer to the eventTime. If 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	Float	(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	Float	(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.

Field	Type	Description
meanValue	Float	(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	Float	(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	Float	(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	Float	(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.
uom	UOM	(Optional) 2- or 3-character code for a physical unit specified in the Common Code column of UN/CEFACT Recommendation 20. If present in QuantityElement, defines a unit of measure by which the specified quantity is to be interpreted as a physical measure. uom SHALL be omitted if quantity is omitted. If present in SensorReport, 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". *Note: GS1 provides a GitHub Repository containing a machine-readable file which enables the automatic conversion between quantitative values expressed using UN/CEFACT common codes. [include URL once the file has been moved into GS1 GitHub Rep]*
coordinateReferenceSstem	anyURI	(Optional) A URI identifying the Coordinate Reference System. **If omitted, the World Geodetic System 1984 (WGS-84) is assumed to apply.**

1030

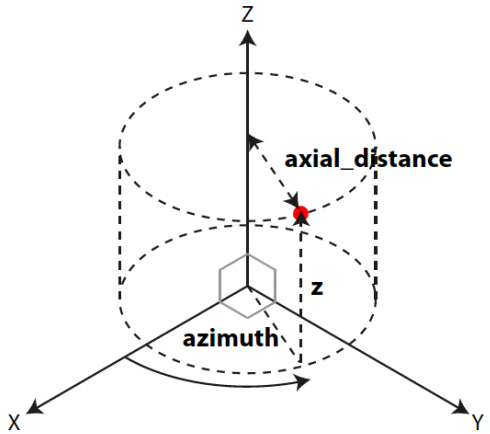
7.3.7.1.2.1 Coordinate reference systems (CRS)

Cartesian coordinates
(x, y, z)



x = component parallel to X axis
y = component parallel to Y axis
z = component parallel to Z axis

Cylindrical polar coordinates
(axial_distance, azimuth, z)

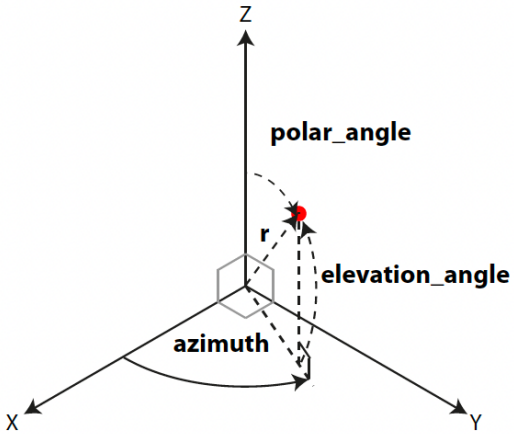


axial_distance = cylindrical radius from z axis
azimuth = angle in XY plane, measured anti-clockwise from the X axis
z = component parallel to Z axis

$$x = axial\_distance \cdot \cos(azimuth)$$
$$y = axial\_distance \cdot \sin(azimuth)$$

$$axial\_distance = \sqrt{x \cdot x + y \cdot y}$$
$$\tan(azimuth) = y / x$$

Spherical polar coordinates
(r, azimuth, polar angle) OR
(r, azimuth, elevation_angle)



r = spherical radius from origin
azimuth = angle in XY plane, measured anti-clockwise from the X axis
polar_angle = angle from Z axis to radius
elevation_angle = angle from XY plane to radius

$$x = r \cdot \cos(azimuth) \cdot \sin(polar\_angle)$$
$$y = r \cdot \sin(azimuth) \cdot \sin(polar\_angle)$$
$$z = r \cdot \cos(polar\_angle)$$
$$z = r \cdot \sin(elevation\_angle)$$

$$r = \sqrt{x \cdot x + y \cdot y + z \cdot z}$$
$$\tan(azimuth) = y / x$$
$$\cos(polar\_angle) = \sin(elevation\_angle) = z / r$$

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].

7.3.7.1.3 Sensor property type

`SensorPropertyTypeID` is a *standard vocabulary* type (see secton 7.2.4) whose elements denote specific sensor properties. An example is an identifier that denotes 'temperature'.

7.3.7.1.4 UOM

The `uom` field qualifies the unit of measure of one or several float 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).

7.3.7.1.5 Microorganism ID

`MicroorganismID` is a *user vocabulary* type (see secton 7.2.4) whose elements denote specific microorganisms. An example is an identifier that denotes 'Listeria monocytogenes'.

7.3.7.1.6 Chemical Substance ID

`ChemicalSubstanceID` is a *user vocabulary* type (see secton 7.2.4) whose elements denote specific chemical substances. An example is an identifier that denotes "sucrose".

7.3.7.1.7 Resource ID

`ResourceID` is a *user vocabulary* type (see secton 7.2.4) 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).

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` (§ *7.4.1.2*), and in `TransformationEvent`s (§ *0*). 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 (§ *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 § *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.

Field	Type	Description
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 § 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).
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 Comprehensive Business Vocabulary standard [CBV2.0] specifies the option of a UUID URI or NI Hash URI 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 § 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 § 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 § 7.3 to be a "date and time in a time zone-independent manner." For example, in the XML binding (§ 9.5) the eventTime field is represented as an element of type xsd:dateTimeStamp, and § 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 modeled by breaking it down into a series of steps, and modeling 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 § *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 modeled by creating suitable EPCIS events. In most situations, this is done using EPCIS events from the Core Event Types Module as specified in § *7.4.1* through *0*, 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 § *7.4.1* through *0*).

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 § *0* 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 modeling 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 § *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).

Fields:

Field	Type	Description

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

`epcList`	`List<EPC>`	(Optional) An unordered list of one or more EPCs naming specific objects to which the event pertained. See § ***Error! Reference source not found.*** 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.

Field	Type	Description
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 associated with the EPCs, presumed to hold true until contradicted by a subsequent event.
persistentDisposition	DispositionID	(Optional) One or more business conditions of the objects associated with the EPCs. 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		(Optional, multivalued) If used in PersistentDisposition, specifies Disposition (cbv:Disp) values to be **set "persistently"**, i.e., **until they are explicitly unset**.
unset		(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 (§ 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 (§ 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 (§ 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 sensorElements (§ 7.3.7).

Note that in the XML binding (§ 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` is unknown.
- (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," meaning an association 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. 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 associations 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 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. 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, and (b) a second `AggregationEvent` with `action` equal to `ADD` at the time the SSCC is assigned (the first association may also be invalidated via a `AggregationEvent` with `action` equal to `DELETE` at this time). The first

AggregationEvent would use the LPN as the parent identifier (expressed in a suitable URI representation; see § 6.4), while the second AggregationEvent would use the SSCC (which is a type of EPC) as the parent identifier, thereby *changing* the parentID.

An AggregationEvent has the following fields:

Field	Type	Description
eventTime recordTime eventTimeZoneOffset	(Inherited from EPCISEvent; see § 7.4.1)	
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 § ***Error! Reference source not found.***. 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 § ***Error! Reference source not found.***. 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 associated with the EPCs, presumed to hold true until contradicted by a subsequent event.
persistentDisposition	DispositionID	(Optional) One or more business conditions of the objects associated with the EPCs. 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).
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.

Field	Type	Description
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 (§ 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 (§ 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 (§ 7.3.7).

Note that in the XML binding (§ 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 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.

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 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 a specific `persistentDisposition` is specified as `set`) The persistent business condition(s) of the objects identified in `parentID`, `childEPCs`, and `childQuantityList` is as set by `persistentDisposition`.
- (If `persistentDisposition` is omitted) The persistent business condition(s) of the objects identified in `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 in `parentID`, `childEPCs`, and `childQuantityList` is unset (i.e., revoked).
- (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` is unknown.
- (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 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).
`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 § 7.4.1)	

`bizTransactionList`	Unordered list of one or more `BusinessTransaction` instances	The business transaction(s).

Field	Type	Description
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 § ***Error! Reference source not found.*** 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).
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 associated with the objects, presumed to hold true until contradicted by a subsequent event.
persistentDisposition	DispositionID	(Optional) One or more business conditions of the objects associated with the EPCs. 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).
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 (§ 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 (§ 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 (§ 7.3.7).

Note that in the XML binding (§ 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` (§ 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 § 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 a specific `persistentDisposition` is specified as `set`) The persistent business condition(s) of the objects in `epcList` and `quantityList` and containing entity `parentID` (if specified) is as set by `persistentDisposition`.
- (If `persistentDisposition` is omitted) The persistent business condition(s) of the objects in `epcList` and `quantityList` and containing entity `parentID` (if specified) 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 in `epcList` and `quantityList` and containing entity `parentID` (if specified) is unset (i.e., revoked).
- (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) is unknown.

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 § 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. See § ***Error! Reference source not found.***.
`inputQuantityList`	`List<Quantity Element>`	(Optional) An unordered list of one or more `QuantityElements` 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. See § ***Error! Reference source not found.***.
`outputQuantityList`	`List<Quantity Element>`	(Optional) An unordered list of one or more `QuantityElements` 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.
`disposition`	`DispositionID`	(Optional) The business condition of the objects associated with the output objects, presumed to hold true until contradicted by a subsequent event.

Field	Type	Description
persistentDisposition	DispositionID	(Optional) One or more business conditions of the objects associated with the EPCs. 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).
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 (§ 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 (§ 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 (§ 7.3.8) that describes the output objects created during this event.
sensorElementList	List<sensorElement>	(Optional) An unordered list of one or more sensorElements (§ 7.3.7).

Note that in the XML binding (§ 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` is unknown.
- (If `ilmd` is non-empty) The objects identified by `outputEPCList` and `outputQuantityList` are described by the attributes in `ilmd`.

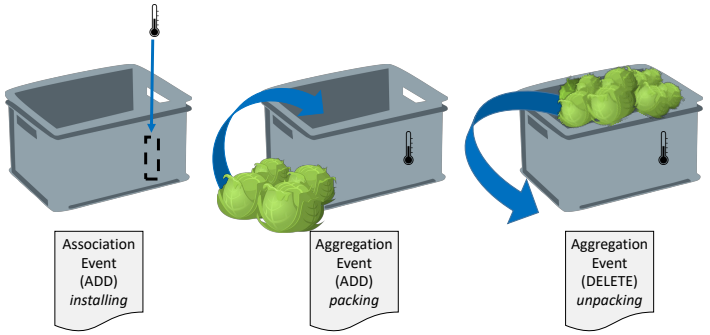
7.4.6 AssociationEvent (subclass of EPCISEvent)

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

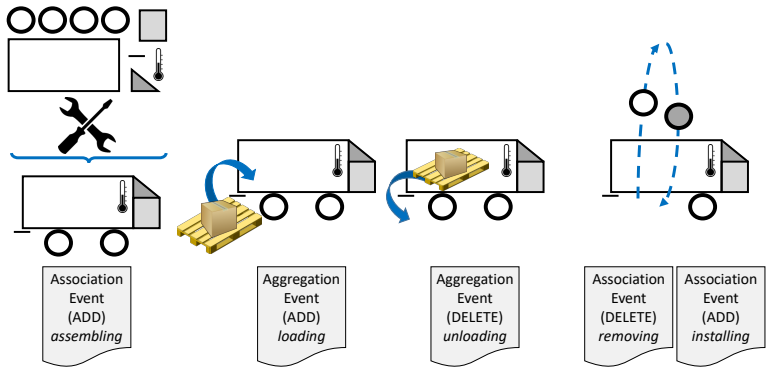
Like the `AggregationEvent`, the `AssociationEvent` is also used to capture associations where there is a strong physical relationship between the containing and the contained objects such that they will all occupy the same location at the same time, until such time as they are disaggregated. However, 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:



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

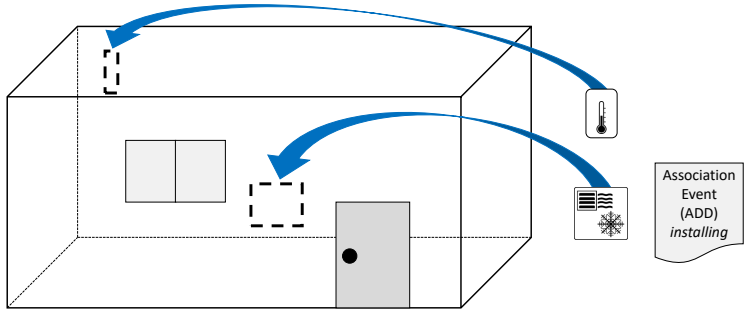


`AggregationEvent` with `action` DELETE while not indicating any of the aggregated children, it would mean that ALL children ever aggregated to that parent ID are no longer aggregated to it – i.e. not just the pallets, the maritime containers or the vegetables, but also all assemblies, screws, tyres, 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.

The three examples to the right 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 § 7.3.)	
parentID	URI	(Optional when action is OBSERVE, required otherwise) 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].
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 § ***Error! Reference source not found.***. 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 objects associated with the EPCs, presumed to hold true until contradicted by a subsequent event.
persistentDisposition	DispositionID	(Optional) One or more business conditions of the objects associated with the EPCs. 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).
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 (§ 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 (§ 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 (§ 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 a specific `persistentDisposition` is specified as `set`) The persistent business condition(s) of the objects in `parentID`, `childEPCs`, and `childQuantityList` is as set by `persistentDisposition`.
- (If `persistentDisposition` is omitted) The persistent business condition(s) of the objects in `parentID`, `childEPCs`, and `childQuantityList` 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 id in `parentID`, `childEPCs`, and `childQuantityList` is unset (i.e., revoked).
- (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` is unknown.
- (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

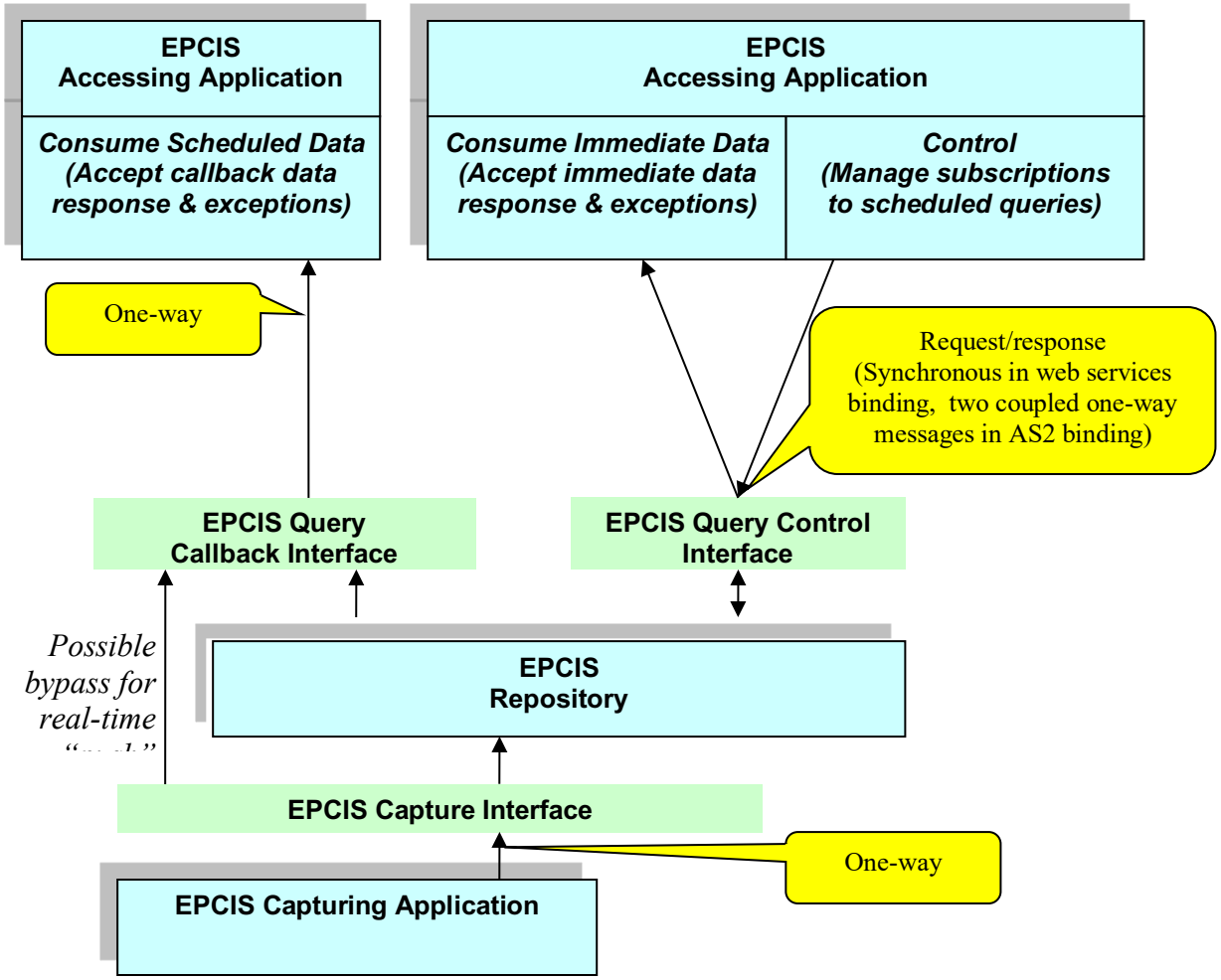
the `bizTransactionList` and all same field values as the `AssociationEvent`. Note, however, that an `AssociationEvent` with a non-empty `bizTransactionList` does not cause a `TransactionEvent` to be returned from a query.

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.

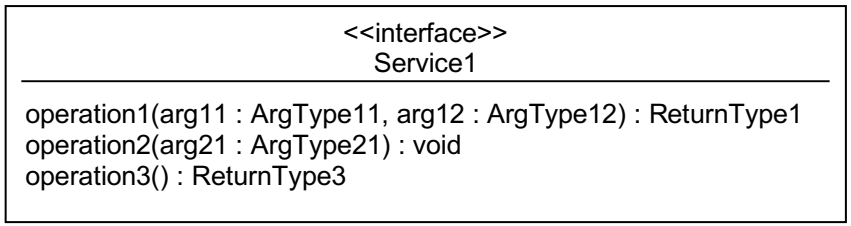
8 Service layer

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 to the right illustrates the relationship between these interfaces, expanding upon the diagram in § 2 (this diagram is non-normative).



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

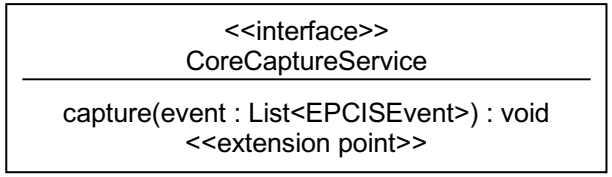
Some bindings of the EPCIS Capture Interface provide a means for the EPCIS Service to authenticate the client's identity, for the client to authenticate the EPCIS Service's identity, or both. The specification of the means to authenticate is included in the specification of each binding. If the EPCIS 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 § *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 and returns no results. 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. Explanation (non-normative): this treatment of `recordTime` is necessary in order for standing queries to be processed properly. See § 8.2.5.2.

Return value:

(none)

The concrete bindings of the EPCIS Capture Interface in § 11 use the EPCIS document structure defined in § 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. Such master data SHALL be queryable by using the query parameters of the SimpleEventQuery specified in § 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 § 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

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 § *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, cursoring, etc. Nothing more complicated was agreed to in this version for the sake of expedience.

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 (§ 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 (§ 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 § 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 § 8.2.7.
getSubscriptionIDs	Returns a list of all subscriptionIDs 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.
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, it SHALL be interpreted as though that query parameter were omitted altogether.

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 § 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`.

Argument	Type	Description
`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 § *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.

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 § *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`	(Not implemented in EPCIS 1.2) The specified query name already exists.
`QueryValidationException`	(Not implemented in EPCIS 1.2) The specified query is invalid; e.g., it contains a syntax error.

Exception Name	Meaning
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.
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.

1866

1867 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

EPCIS Method	Exceptions
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 `eventTypes` 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 (§ 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 NEED NOT 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
`eventTypes`	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`, `QuantityEvent`, `TransactionEvent`, or `TransformationEvent`. An 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.

Parameter name	Parameter value type	Meaning
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).
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 (§ 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 (§ 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 String	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 String	Like the EQ_bizStep parameter, but for the disposition field.
EQ_persistentDisposition_set	List of String	Like the EQ_bizStep parameter, but for the persistentDisposition set field.
EQ_persistentDisposition_unset	List of String	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_INNER_readPoint_*fieldname*	List of String	Analogous to EQ_INNER_readPoint_*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).
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 § 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.
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).
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 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) 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 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) 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 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) 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 String	If this parameter is specified, the result will only include events that (a) have a transformationID field (that is, TransformationEvents or extension event type that extend TransformationEvent); and where (b) the transformationID field is equal to one of the values specified in this parameter.
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 or extension event types that extend one of those three); 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 § 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.

Parameter name	Parameter value type	Meaning
MATCH_parentID	List of URIs	Like MATCH_epc, but matches the parentID field of AggregationEvent, the parentID field of TransactionEvent, and extension event types that extend either AggregationEvent or TransactionEvent. The meaning of "matches" is as specified in § 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 § 8.2.7.1.1. If this parameter is omitted, events are included regardless of their inputEPCList field or whether the inputEPCList field exists.
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 § 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, or extension event types that extend one of those four); 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 § 8.2.7.1.1.
MATCH_epcClass	List of String	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 or extension event types that extend one of those three); 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 § 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 § 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 § 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, or extension event types that extend one of those four); 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 § 8.2.7.1.1.
EQ_quantity	Int	(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.
EQ_quantity_*uom*	List of String	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	Int	(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*	List of String	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.

Parameter name	Parameter value type	Meaning
GE_quantity	Int	(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*	List of String	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	Int	(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*	List of String	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	Int	(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.
LE_quantity_*uom*	List of String	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.

Parameter name	Parameter value type	Meaning
EQ_*fieldname*	Int Float DateTimeStamp	Like EQ_*fieldname* as described above, but may be applied to a field of type Int, Float, 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, or Time); and where (c) the value of the field is equal to the specified value. *fieldname* is constructed as for EQ_*fieldname*.
GT_*fieldname*	Int Float DateTimeStamp	Like EQ_*fieldname* as described above, but may be applied to a field of type Int, Float, 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, 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	Analogous to GT_*fieldname*
EQ_ILMD_*fieldname*	List of String	Analogous to EQ_*fieldname*, but matches events whose ILMD area (§ 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.
GT_ILMD_*fieldname* GE_ILMD_*fieldname* LT_ILMD_*fieldname* LE_ILMD_*fieldname*	Int Float DateTimeStamp	Analogous to EQ_*fieldname*, GT_*fieldname*, GE_*fieldname*, GE_*fieldname*, LT_*fieldname*, and LE_*fieldname*, respectively, but matches events whose ILMD (§ 7.3.8) contains a field having the specified *fieldname* whose integer, 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.

Parameter name	Parameter value type	Meaning
GT_INNER_*fieldname* GE_INNER_*fieldname* LT_INNER_*fieldname* LE_INNER_*fieldname*	Int Float DateTimeStamp	Like EQ_INNER_*fieldname* as described above, but may be applied to a field of type Int, Float, or Time.
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	Like EQ_INNER_ILMD_*fieldname* as described above, but may be applied to a field of type Int, Float, 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 (§ 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.
EQATTR_*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_eventID	List of String	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.

Parameter name	Parameter value type	Meaning
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 String	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.
EQ_correctiveEventID	List of String	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.
GT_ERROR_DECLARATION_*fieldname* GE_ERROR_DECLARATION_*fieldname* LT_ERROR_DECLARATION_*fieldname* LE_ERROR_DECLARATION_*fieldname*	Int Float DateTimeStamp	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, 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).
GT_INNER _ERROR_DECLARATION_*fieldname* GE_INNER _ERROR_DECLARATION_*fieldname* LT_INNER _ERROR_DECLARATION_*fieldname* LE_INNER _ERROR_DECLARATION_*fieldname*	Int Float DateTimeStamp	Like EQ_INNER_ERROR_DECLARATION_*fieldname* as described above, but may be applied to a field of type Int, Float, 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.
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, 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.)

Parameter name	Parameter value type	Meaning
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.
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.
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 § 8.2.3).

Parameter name	Parameter value type	Meaning
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).
EQ_type	List of String	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.

Parameter name	Parameter value type	Meaning
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.
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 String	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 Float	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*	Float	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_minValue_*uom*	Float	Like GT_value_*uom*, but pertaining to the minValue attribute.
GE_minValue_*uom*	Float	Like GE_value_*uom*, but pertaining to the minValue attribute.
LT_minValue_*uom*	Float	Like LT_value_*uom*, but pertaining to the minValue attribute.
LE_minValue_*uom*	Float	Like LE_value_*uom*, but pertaining to the minValue attribute.

Parameter name	Parameter value type	Meaning
GE_maxValue_*uom*	Float	Like GE_value_*uom*, but pertaining to the maxValue attribute.
LT_maxValue_*uom*	Float	Like LT_value_*uom*, but pertaining to the maxValue attribute.
GT_meanValue_*uom*	Float	Like GT_value_*uom*, but pertaining to the meanValue attribute.
GE_meanValue_*uom*	Float	Like GE_value_*uom*, but pertaining to the averageValue attribute.
LT_meanValue_*uom*	Float	Like LT_value_*uom*, but pertaining to the averageValue attribute.
LE_meanValue_*uom*	Float	Like LE_value_*uom*, but pertaining to the meanValue attribute.
EQ_stringValue	List of String	If this parameter is specified, the result will only include events that (a) have a valueType and a value attribute; and where (b) a pair of valueType and value is equal to the specified parameter. If omitted, events are in included regardless of the values of their valueType and value attributes.
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.
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.

Parameter name	Parameter value type	Meaning
GT_SENSORELEMENT_*fieldname* GE_SENSORELEMENT_*fieldname* LT_SENSORELEMENT_*fieldname* LE_SENSORELEMENT_*fieldname*	Int Float DateTimeStamp	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, float, 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.
GT_INNER_SENSORELEMENT_*fieldname* GE_INNER_SENSORELEMENT_*fieldname* LT_INNER_SENSORELEMENT_*fieldname* LE_INNER_SENSORELEMENT_*fieldname*	Int Float DateTimeStamp	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, float, or time value matches the specified value according to the specified relational operator.
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.
GT_SENSORMETADATA_*fieldname* GE_SENSORMETADATA_*fieldname* LT_SENSORMETADATA_*fieldname* LE_SENSORMETADATA_*fieldname*	Int Float DateTimeStamp	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, float, 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
GT_INNER_SENSORMETADATA_*fieldname* GE_INNER_SENSORMETADATA_*fieldname* LT_INNER_SENSORMETADATA_*fieldname* LE_INNER_SENSORMETADATA_*fieldname*	Int Float DateTimeStamp	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, float, 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.
GT_SENSORREPORT_*fieldname* GE_SENSORREPORT_*fieldname* LT_SENSORREPORT_*fieldname* LE_SENSORREPORT_*fieldname*	Int Float DateTimeStamp	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, float, 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.
GT_INNER_SENSOREPORT_*fieldname* GE_INNER_SENSOREPORT_*fieldname* LT_INNER_SENSOREPORT_*fieldname* LE_INNER_SENSOREPORT_*fieldname*	Int Float DateTimeStamp	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, float, or time value matches the specified value according to the specified relational operator.
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.

Parameter name	Parameter value type	Meaning
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_sDev	Float	If this parameter is specified, the result will only include events that (a) have a sDev attribute; and where (b) a value of sDev is greater than or equal to the specified parameter.
LT_sDev	Float	If this parameter is specified, the result will only include events that (a) have an sDev attribute; and where (b) a value of sDev is less than the specified parameter.
GE_percRank	Float	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.
LT_percRank	Float	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.
GE_percValue	Float	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.
LT_percValue	Float	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.

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 pattern, as specified in § *7.3.3.1*). 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:0614141.112345.*, then this event would be matched by the query parameter urn:epc:idpat:sgtin:0614141.*.* or by urn:epc:idpat:sgtin:0614141.112345.*, but not by urn:epc:idpat:sgtin:0614141.112345.400.

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 § 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:1`) 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:1`) 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.

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 (§ 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.

EventTypes defined in EPCIS 2.0 (i.e., `AssociationEvent`) appear within **two `<extension>` elements** 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.

Additional fields added to EPCIS 2.0 (e.g., `sensorElement`) appear **within a second `<extension>` element** that is nested within the first `<extension>` element, following the EPCIS 1.2 fields.

If additional fields are added in a future version of EPCIS, they will appear within **a third `<extension>` element** that is nested within the first `<extension>` element, following the EPCIS 2.0 fields.

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 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.

See external xml and json artifacts for non-normative 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.

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`

SBDH Field (XPath)	Value
`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`
`EPCISMasterData` used in any context	`MasterData`
`EPCISQueryDocument` used as the request side of the binding in § *11.3*	`QueryControl-Request`
`EPCISQueryDocument` used as the response side of the binding in § *11.3*	`QueryControl-Response`
`EPCISQueryDocument` used in any XML binding of the Query Callback interface (§ *11.4.2 – 11.4.4*)	`QueryCallback`
`EPCISQueryDocument` used in any other context	`Query`

The AS2 binding for the Query Control Interface (§ *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 § *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, make reference to the EPCglobal Base Schema. This schema can be found in the external artefact, "EPCglobal.xsd".

9.4 Master data in the XML binding

As noted in § *0*, EPCIS provides two ways to transmit master data. These four ways are 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

Mechanism	Schema Support
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 four 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:1"
    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 limted to FIT mappings) are tolerated in XML validaton by lax processing of the `##other` namespace.

9.5 Schema for core event types

The Core Event Types data definition module, an external XML Schema (XSD) artefact, "EPCglobal-epcis-2_0.xsd", imports additional schemas as shown in the following table:

Namespace	Location Reference	Source
`urn:epcglobal:xsd:1`	EPCglobal.xsd	§ *9.3*
`http://www.unece.org/cefact/namespaces/StandardBusinessDocumentHeader`	StandardBusinessDocumentHeader.xsd	UN/CEFACT web site; see § *9.2*

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 (a) for returning results from the `SimpleMasterDataQuery` query type (§ *8.2.7.2*); (b) to provide the body of a master data document as defined in § ***Error! Reference source not found.***; and (c) 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 § 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 can be found in the external artefact, "EPCglobal-epcis-2_0.xsd".

9.6 Core event types – examples (Non-Normative)

Non-normative examples can be found in external xml and json artifacts.

10 JSON / JSON-LD bindings for data definition

This chapter defines the JSON / JSON-LD data binding 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 binding for EPCIS that is defined in § 9.

§ 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.

§ 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.

§ 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.

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

§ 10.5 provides references to external standards introduced in this chapter.

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 binding in JSON and JSON-LD format as an alternative to XML, which was the only data binding specified in EPCIS 1.2 and earlier. XML continues to be supported in EPCIS v2.0.

The additional JSON/JSON-LD data binding 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)

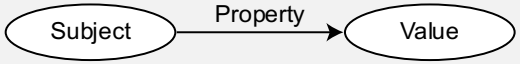
About Linked Data

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 technology is also referred to as Semantic Web technology.
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).



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://tools.ietf.org/html/rfc8259] and as ISO/IEC 21778:2017. 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 [*https://www.w3.org/TR/json-ld/*] 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), etc.

Because EPCIS 1.2 already provides an XML data binding 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 binding.

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 object 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 object 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 object (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.

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.
"@type"	"@type" is used to specify that a string value should be cast to another data type, such as an xsd:dateTimeStamp.
"@vocab"	"@vocab" is used to specify the default vocabulary, i.e. the namespace that is assumed for all unqualified fieldnames that are not already full URIs or Compact URI Expressions (CURIEs). In the JSON-LD binding for EPCIS 2.0, the default vocabulary is set to the same URI stem as for the epcis: CURIE prefix. **https://ns.gs1.org/epcis/** has been allocated as **namespace** for this purpose.
"@context"	"@context" is used to specify the JSON-LD context object, either declared explicitly inline or via URL reference. See § 10.1.3 for further details

Many users of EPCIS 2.0 will initially focus on the JSON representation and may disregard the context object. However, retrieval and usage of the context object 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 object to express namespace mappings, aliases and to cast values of particular data fields to specific data types. Features of the JSON-LD context object that are relevant for EPCIS 2.0 are explained in § 10.1.3.

It should be noted that a JSON-LD dataset may reference or include more than one context object. Typically it may reference **the standard JSON-LD context object for EPCIS 2.0, which is hosted at https://ns.gs1.org/ref/EPCIS-context**. It ma also reference one or more additional JSON-LD context objects 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 object

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

- Default vocabulary (specified using "@vocab")

- Expansions for Compact URI Expression (CURIE) prefixes,
  e.g. "xsd": "http://www.w3.org/2001/XMLSchema#"
  In this sense, the context header 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 the context header, e.g. "example": "http://ns.example.com/epcis/"
- Mappings of fields to data types other than string (type casting), e.g.:
    - `"eventTime": {"@type":"xsd:dateTimeStamp"}`
      Every value for eventTime (and also recordTime and declarationTime) should be treated as an xsd:dateTimeStamp value
    - `"quantity": {"@type":"xsd:decimal"}`
      Every value of quantity should be treated as an xsd:decimal value
    - `"epcList": {"@type":"@id"}`
      Every value within epcList (and also epcClass, parentID, childEPCs, inputEPCList, outputEPCList, bizStep, disposition, readPoint, bizLocation, bizTransaction, source, destination, type) should be treated as URIs (xsd:anyURI) – although JSON-LD uses the special notation "@id" for this purpose.
- Aliases for JSON-LD keywords, e.g.:
    - `"isA": "@type"`
    - `"id":"@id"`
    - Mappings of EPCIS fields to semantic URIs defined elsewhere, e.g.:
        - `"creationDate": {"@id":"dcterms:created", "@type":"xsd:dateTimeStamp"}`
        - `"format": {"@id":"dcterms:format", "@type":"xsd:string"}`
        - `"schemaVersion": {"@id":"owl:versionInfo", "@type":"xsd:string"}`
          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.

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 object 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 header. 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 header 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 binding which is validated using XML Schema Definition language (XSD). In addition to the XML data binding (which is updated in EPCIS v2.0 to support the new AssociationEvent and SensorElement), EPCIS v2.0 defines a JSON / JSON-LD data binding.

JSON data can be validated using JSON Schema [*https://json-schema.org/specification.html*]. EPCIS 2.0 requires at least v7 of JSON Schema because it makes use of the if/then feature introduced in JSON Schema v7.

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.

Data format	Validation Language
XML	XSD
JSON	JSON Schema
JSON-LD	SHACL

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 as between the opening 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 indicated by enclosing 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 JSON-LD context object, `"action": {"@type":"xsd:string"}`

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 JSON-LD context object, `"eventTime": {"@type":"xsd:dateTimeStamp"}`

Format	EPCIS 2.0 example for bizStep (same pattern for disposition, eventID)
XML	`<bizStep>urn:epcglobal:cbv:bizstep:shipping</bizStep>`
JSON	`{"bizStep": "urn:epcglobal:cbv:bizstep:shipping"}`
JSON-LD	
	`Within JSON-LD context object, "bizStep": {"@type":"@id"}`

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 object. Note also that for bizStep, disposition and other fields that expect a URI value, the context object indicates "@type":"@id" rather than "@type":"xsd:anyURI". This is a particular quirk of JSON-LD.

10.2.2 Validating data fields expecting simple values

For simple values that are not an xsd:string, the JSON-LD context object specifies the @type for each field within an EPCIS event. Note that where the simple value is a URI (xsd:anyURI), JSON-LD indicates this via the special key:value pair "@type":"@id" within the JSON-LD context object, "@id" indicating that the value should be considered as an IRI or URI.

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 § 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" />`

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

Format	EPCIS 2.0 validation example for eventTime (similar pattern for recordTime)
SHACL	epcis:EventTime_TypeConstraint owl:sameAs [sh:path epcis:eventTime ; sh:datatype xsd:dateTimeStamp ; sh:name "eventTime type constraint" ; sh:message "eventTime must be an xsd:dateTimeStamp"]. epcis:EventTime_IsMandatory owl:sameAs [sh:path epcis:eventTime ; sh:name "eventTime is mandatory" ; sh:minCount 1; sh:maxCount 1; sh:message "eventTime is a mandatory field which must have a single value."].

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 expressed 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>`

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

SHACL	`epcis:BizStep_TypeConstraint owl:sameAs [`
	`sh:path epcis:bizStep ;`
	`sh:nodeKind sh:IRI ;`
	`sh:name "bizStep type constraint" ;`
	`sh:message "bizStep must be an xsd:anyURI. Please refer to the GS1 Comprehensive Business Vocabulary for standard vocabulary values for bizStep"`
	`].`

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 references a definition named `uri`, in which the "`type`" attribute specifies "`string`" and the "`format`" attribute specifies "`uri`".

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 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:Action_TypeAndFormat owl:sameAs [`
	`sh:path epcis:action ;`
	`sh:datatype xsd:string ;`
	`sh:name "action" ;`
	`sh:in ("ADD" "OBSERVE" "DELETE");`
	`sh:message "action must have an upper case string value - 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 value of the "`enum`" property.

In SHACL, the same enumerated strings are within the list value 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, 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 binding, 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 § 10.2.6.

Format	EPCIS 2.0 example for epcList

XML	`<epcList>`
	`  <epc>urn:epc:id:sgtin:0614141.107346.2017</epc>`
	`  <epc>urn:epc:id:sgtin:0614141.107346.2018</epc>`
	`</epcList>`

JSON	`{"epcList": ["urn:epc:id:sgtin:0614141.107346.2017","urn:epc:id:sgtin:0614141.107346.2018"]}`
JSON-LD	Within JSON-LD context object, `"epcList": {"@type":"@id"}` because the EPC values are URIs

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>`

Format	EPCIS 2.0 validation example for epcList

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

SHACL	`epcis:EPCList_TypeConstraint  owl:sameAs [`
	`  sh:path epcis:epcList ;`
	`  sh:nodeKind sh:IRI ;`
	`  sh:name "epcList type constraint" ;`
	`  sh:message "epcList must be a list of xsd:anyURI. "`
	`].`

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.

In SHACL, there is no need to declare that a list of values is expected.

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 bare child string values and inline attributes are treated equally, with the name of the XML child element being the key in JSON / JSON-LD whose value was the bare child string value in XML. The inline attribute (typically named `'type'`) and its value are expressed as another key:value pair within the same JSON data object.

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:0614141073467:1152 </bizTransaction> </bizTransactionList>
JSON JSON-LD	{"bizTransactionList": [{"type":"urn:epcglobal:cbv:btt:po", "bizTransaction": "http://transaction.acme.com/po/12345678" }, {"type":"urn:epcglobal:cbv:btt:desadv", "bizTransaction": "urn:epcglobal:cbv:bt:0614141073467:1152" }]}
	Within JSON-LD context object, "type": {"@type":"@id"} "bizTransaction": {"@type":"@id"} because the values of 'type' and 'bizTransaction' are both URIs

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) 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:ObjectEventTypes" 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="AggregationEventTypes">
  <xsd:complexContent>
    <xsd:extension base="epcis:EPCISEventTypes">
        <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`.

EPCIS 2.0 defines an alias of the special JSON-LD keyword "`@type`" named "`isA`" 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. The "`isA`" alias for "`@type`" is declared in the standard JSON-LD context header for EPCIS 2.0.

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 `'isA' (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 "isA" 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 explcitly. However, in EPCIS 2.0 event data, there are no such declarations *within the EPCIS data* that an epcis:ObjectEvent is an rdfs:subClassOf epcis:Event , nor can this subclass relationship be defined within the JSON-LD context object.

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 and such validation rules on core fields of `EPCISEvent` are simply replicated for each EPCIS event type.

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*

Validation Rule	XSD	JSON Schema	SHACL
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`)
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 floating-point value	type="`xsd:double`"	`"type":"number"`	`sh:datatype xsd:double`
Field expects a URI	type="`xsd:anyURI`" or references an `xsd:simpleType` with an `xsd:restriction base="xsd:anyURI"`	`"type":"string",` `"format":"uri"`	`sh:nodeKind sh:IRI`

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:Action_TypeAndFormat owl:sameAs [
			sh:path epcis:action ;
			sh:datatype xsd:string ;
			sh:name "action" ;
			sh:in epcis:ActionEnum ;
			sh:message "action must have an upper case string value - either ADD, OBSERVE or DELETE"
			].
			
			epcis:ActionEnum owl:sameAs ("ADD" "OBSERVE" "DELETE").

10.2.9 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.org/playground/
- http://rdfshape.herokuapp.com/

10.2.10 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 provided at:
https://github.com/gs1/EPCIS/blob/master/JSON/EPCIS-JSON-Schema.json

***!!! Need to update this link before ratification to the stable URL hosted at gs1.org

The JSON-LD representation of EPCIS 2.0 data SHALL validate against the SHACL file provided at:

https://github.com/gs1/EPCIS/blob/master/JSON/EPCIS-SHACL.ttl

***!!! Need to update this link before ratification to the stable URL hosted at gs1.org

§ 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 binding. 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 hosted at https://ns.gs1.org/ref/ .

10.4 Non-normative examples in JSON and JSON-LD

JSON / JSON-LD examples equivalent to the non-normative XML examples referenced in § 9 are provided in external artifacts.

JSON / JSON-LD examples for sensor data are provided in external artifacts.

JSON / JSON-LD examples for the AssociationEvent are provided in external artifacts.

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 § 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 an XML 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 § 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 § 9.5; or
- an XML document whose root element conforms to the `EPCISQueryDocument` element as defined by the schema of § 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.

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 § 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 § *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 § *9.5*; or
- an XML document whose root element conforms to the `EPCISQueryDocument` element as defined by the schema of § *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], § *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

Code examples: This chapter contains OpenAPI 3.0 examples in YAML throughout the document, which is easier to read than JSON. All examples can be converted to a JSON representation. Furthermore, **the OpenAPI specifications in YAML and JSON will be available at https://ns.gs1.org/ref/** .

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, authorization, 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
/	X			
/capture	X	X	X	
/capture/{captureID}	X	X		
/eventTypes	X	X		
/eventTypes/{eventType}	X	X		
/eventTypes/{eventType}/events	X	X	X	
/eventTypes/{eventType}/events/{eventID}	X	X		
/epcs	X	X		
/epcs/{epc}	X	X		
/epcs/{epc}/events	X	X		
/epcs/{epc}/events/{eventID}	X	X		
/bizSteps	X	X		
/bizSteps/{bizStep}	X	X		
/bizSteps/{bizStep}/events	X	X		
/bizSteps/{bizStep}/events/{eventID}	X	X		
/bizLocations	X	X		
/bizLocations/{bizLocation}	X	X		
/bizLocations/{bizLocation}/events	X	X		
/bizLocations/{bizLocation}/events/{eventID}	X	X		

/readPoints	X	X		
/readPoints/{readPoint}	X	X		
/readPoints/{readPoint}/events	X	X		
/readPoints/{readPoint}/events/{eventID}	X	X		
/dispositions	X	X		
/dispositions/{disposition}	X	X		
/dispositions/{disposition}/events	X	X		
/dispositions/{disposition}/events/{eventID}	X	X		
/queries	X	X		
/queries/{queryName}	X	X	X	X
/queries/{queryName}/subscriptions	X	X	X	
/queries/{queryName}/subscriptions/{subscriptionId}	X	X		X
/queries/{queryName}/events	X	X		
/queries/SimpleEventQuery	X		X	
/queries/SimpleEventQuery/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 § 13.

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 `415 Unsupported Media Type`. 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, a server SHALL support `GS1-CBV-Version`, `GS1-EPC-Format`, `GS1-CBV-Format`, `GS1-CBV-Extensions`, `GS1-EPCIS-Version`, `GS1-EPCIS-Min` and `GS1-EPCIS-Max`. 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-EPCIS-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

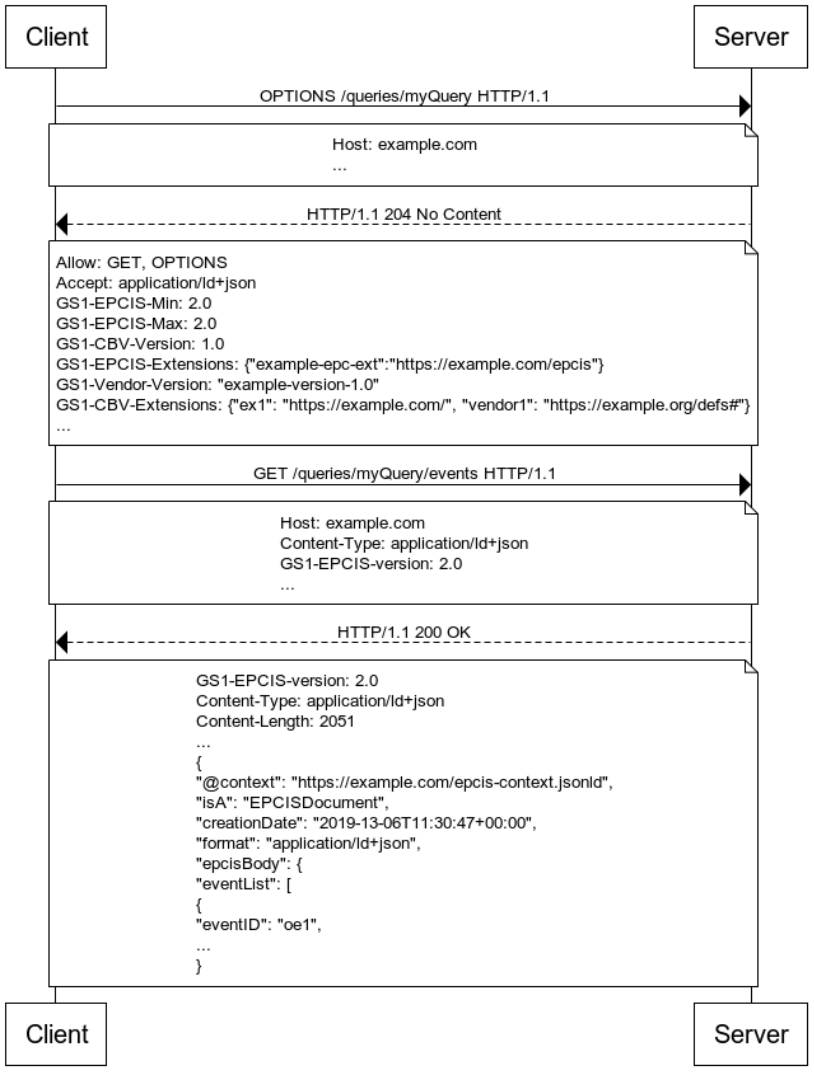
Header	Description	Examples	Request	Response
GS1-EPCIS-Version	The EPCIS version	1.0, 1.1, 1.2, 2.0	X	X
GS1-EPCIS-Min	The lowest EPCIS version supported	>=1.0	X	X
GS1-EPCIS-Max	The highest EPCIS version supported	1.0, 1.1, 1.2, 2.0	X	X
GS1-CBV-Version	The core business vocabulary version	1.2.2, 2.0	X	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-Format	Request or response header to indicate whether CBV URI fields are expressed as URLs or URNs.	Always_URL	X	X
GS1-EPCIS-Extensions	Specific EPCIS extensions supported (e.g. for FIT)	`"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

Header	Description	Examples	Request	Response
GS1-CBV-Extensions	Specific CBV extensions supported (e.g. for fish, rail, FIT). An optional header to link to the top-level resource's vocabulary using the Compact URI Expression [CURIE] syntax	« ex1 »: « http://example.com/ », « vendor1 »: « http://example.org/defs# »	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
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

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.

Figure 12-1 Client first uses OPTIONS to discover which versions are supported and making GET request

The server SHALL support the `OPTIONS` method for each endpoint and SHALL provide default header values for each custom EPCIS header it supports.



12.4 Authentication and Authorization

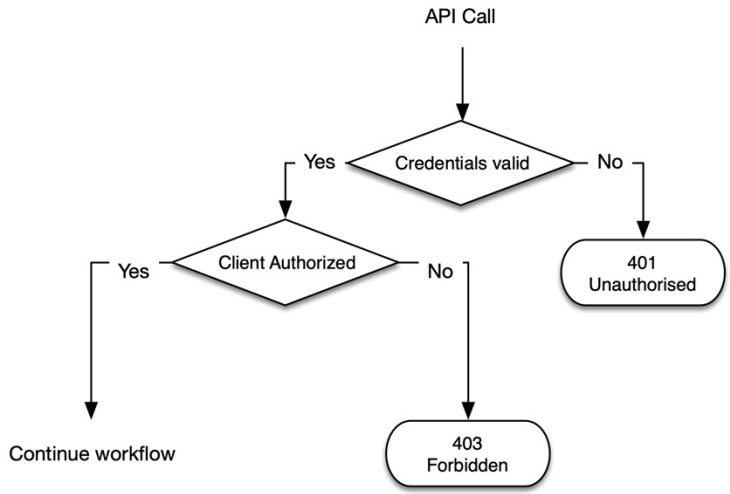


Figure 12-2 Authentication and authorisation

EPCIS clients SHALL authenticate themselves for every API call. An EPCIS server SHOULD implement an authentication mechanism using the `Authorization` header. If the authentication fails, the server SHALL respond with HTTP status code `401` [RFC7235].

Authorized

Request:
`GET /eventTypes/ObjectEvent/events HTTP/1.1`
`Host: example.com`
`Authorization: MyValidSecretAPIKey`
Response:
`HTTP/1.1 200`

Unauthorized

Request:
`GET /eventTypes/ObjectEvent/events HTTP/1.1`
`Host: example.com`
`Authorization: MyInvalidSecretAPIKey`
Response:
`HTTP/1.1 401 Unauthorized`

If a client is authenticated but is not authorised to perform an operation, the server SHALL respond with HTTP status code `403 Forbidden` [RFC7235].

12.5 Pagination

An EPCIS repository SHALL implement pagination, to return events in manageable chunks. A client MAY request the number of events that are returned at once, using the `perPage` query string parameter. If the client does not specify the `perPage` value, the server SHALL use the default value of `30`.

Pages SHALL form a linked list, using the Web link model [RFC8288]. A server SHALL include the quoted URL of the next page in the Link response header. A link MAY contain a `rel` annotation. The value `next` indicates to the client that there are more pages to be accessed. The absence of the next page means the client has received the URL of the final page.

Example: Pagination

Request

```
GET /events/all?perPage=50 HTTP/1.1
Host: example.com
```

Response

```
HTTP/1.1 200 OK
link: <https://example.com/events/all?perPage=50&nextPageToken=t%3A1550673874978%2Ci%3AU6D7DENAKwM2gQRRwGrataeq>;
rel=”next”
```

A server MAY optionally collobarotate with clients on resource management by declaring how long a pagination token will remain valid in the optional `GS1-Next-Page-Token-Expires` and the server MAY optionally also allow clients to free allocated tokens through a `DELETE` on the `nextPageToken` endpoint.

12.6 Capturing EPCIS Events

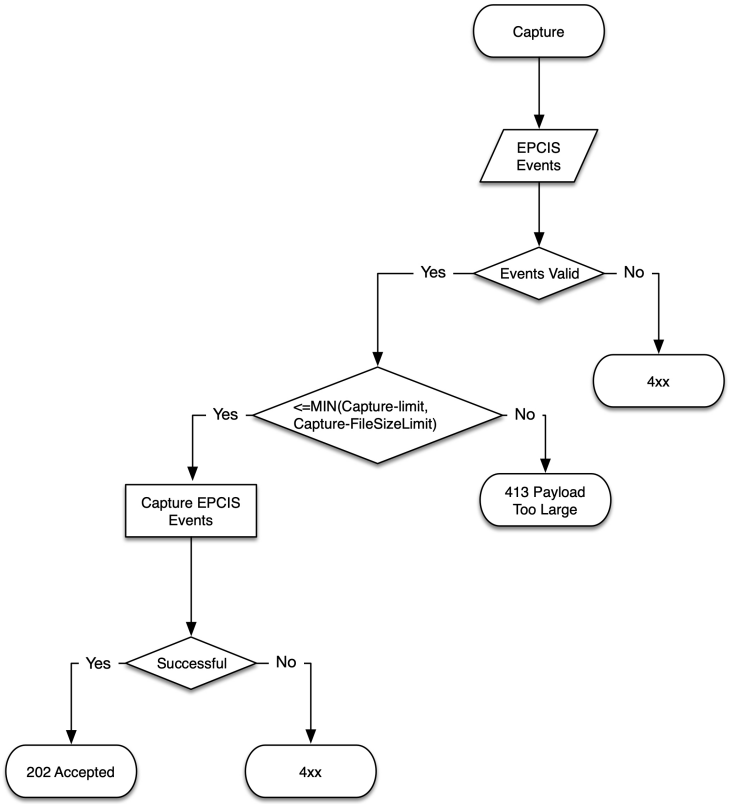
This section defines the RESTful `/capture` endpoint for EPCIS 2.0. Unlike in EPCIS 1.2, the capture interface in EPCIS 2.0 can be asynchronous to allow better scalability for larger sets of events.

Endpoint	GET	POST
/capture	Returns IDs of capture Jobs	Captures one or several EPCIS events. The server may either accept or reject all events if value of `GS1-Capture-Error-Behaviour` header is `rollback` (this is default behaviour). The server may accept all valid events if value of `GS1-Capture-Error-Behaviour` is `proceed`.
/capture/{captureID}	Returns the state of the capture job.	Not supported.

12.6.1 Capture Interface

Client calls the capture interface to send one or more EPCIS events in the JSON or JSON-LD format. If the event syntax is XML, events SHALL conform to the XML event specifications specified in EPCIS 2.0 [EPCIS2.0]. If the event syntax is JSON/JSON-LD, events SHALL conform to the JSON/JSON-LD data bindings specified in § 10.

Figure 12-3 Endpoint: Capture Interface workflow



If `GS1-Capture-Error-Behaviour` is `rollback`, the server SHALL guarantee that either all events are captured, or all events are rejected. If `GS1-Capture-Error-Behaviour` is `rollback`, the server SHALL try to capture as many EPCIS events as possible. If events were successfully accepted, `/capture` SHALL respond with `202 Accepted` and a `captureID`. Upon capture, a server MAY populate the unique eventID field within each event. If unique eventIDs are populated, these SHALL use one of the two eventID formats specified in the Event ID section of the CBV.

Implementations that specify a file size limit may wish to check content length in order to avoid validating files which exceed this limit; this also provides a safeguard against denial-of-service attacks.

If the client sends a payload that exceeds the number of events permitted according to `GS1-EPCIS-Capture-Limit` or the content length exceeds `GS1-EPCIS-Capture-File-Size-Limit`, the server SHALL respond with `413 Payload Too Large` and include the most restrictive capture limit constraint. In other cases of failures, `/capture` SHALL indicate why the operation was unsuccessful by returning a HTTP status code in the range `4xx`.

12.6.2 Capture Jobs Interface

When EPCIS events are added through the capture interface, the capture process runs asynchronously. If the payload is syntactically correct and the client is allowed to call `/capture`, the server returns a `202 Accepted`. This does not guarantee successful storage of all EPCIS events. The capture job endpoint, `/capture/{captureID}`, SHALL expose the state of the capture job to the client.

A capture job document has at least the following properties:

- `running`: whether or not the capture job is still active
- `success`: whether or not at least one error occurred,
- `errors` or `errorFile`: with the errors if success is false. Note that `errorFile` should contain a URL pointing to an error log file.

Only if not a single error occurred, `success` is `true`. If success is false, there was at least one error. If the `GS1-Capture-Error-Behaviour` header is rollback, a `success` value of `false` aborts the capture job and rejects all EPCIS events related to the job. If `GS1-Capture-Error-Behaviour` header is `proceed`, some EPCIS events might still be captured, even if `success` is `false`.

Table 12-3 GS1-Capture-Error-Behaviour header value is `rollback`

Capture job running	Capture job success	Capture job outcome
true	true	Still capturing EPCIS events. No errors occurred so far.
true	false	At least one error occurred. Rollback is in progress.
false	true	All EPCIS events are captured.
false	false	All EPCIS events rejected.

Table 12-4 GS1-Capture-Error-Behaviour header value is `proceed`

Capture job running	Capture job success	Capture job outcome
true	true	Still capturing EPCIS events. No errors occurred so far.

Capture job running	Capture job success	Capture job outcome
true	false	At least one error occurred but more EPCIS events are currently being captured.
false	true	All EPCIS events were captured without an error.
false	false	Some EPCIS events were captured but errors occurred.

If `success` is `false`, check the `errors` or `errorFile` property for details.

12.7 Events interface

This section defines the events endpoint, which provides a RESTful interface to EPCIS events. EPCIS events can be "viewed" from three different aspecs:

- EPCIS event type: EPCIS events are accessed through their EPCIS event type. A special event type is "`all`", which provides a path to events without specifying the event type. Its main purpose is to provide a neutral endpoint (without specifying event type) for queries that potentially retrieve more than one type of event. `/eventTypes/{eventType}/events` is also the only endpoint that supports `POST`.
- Top-level resources: EPCIS events can be accessed based on their context, for example, all EPCIS events at a given location can be accessed through the `/bizLocation` endpoint.
- Query result: EPCIS events can be part of a result of an EPCIS query.

Table 12-5 Overview of EPCIS 2.0 related endpoints

Endpoint	Functionality
`/eventTypes/{eventType}/events`	Returns all EPCIS events of a specific type, if specified. Returns EPCIS event(s) related to the given eventType and eventID. The EPCIS event type 'all' is the union of all EPCIS events and supports accessing events without specifying the event type. Filtering EPCIS events using the EPCIS query language.

	Supports creating a single EPCIS event.

/epcs/{epc}/events
/bizSteps/{bizStep}/events
/bizLocations/{bizLocation}/events

/readPoints/{readPoint}/events /dispositions/{disposition}/events Returns all EPCIS events with a specific top-level resouce. Returns EPCIS event(s) related to the given endpoint (e.g., bizStep, bizLocation, readPoint, disposition) and eventID Filtering EPCIS events using the EPCIS query language.	
/queries/{queryName}/events /queries/SimpleEventQuery/events	Returns EPCIS events that match the query.

12.7.1 EPCIS events collections

When calling the `GET` method on the `/eventTypes` endpoint, it SHALL return a list of all EPCIS event types that are present in that repository.

A server SHALL implement the virtual event type "`all`" as a collection which contains all EPCIS events in the server's repository regardless of their types: following design best practices of RESTful APIs, events SHALL be logically grouped by EPCIS event types.

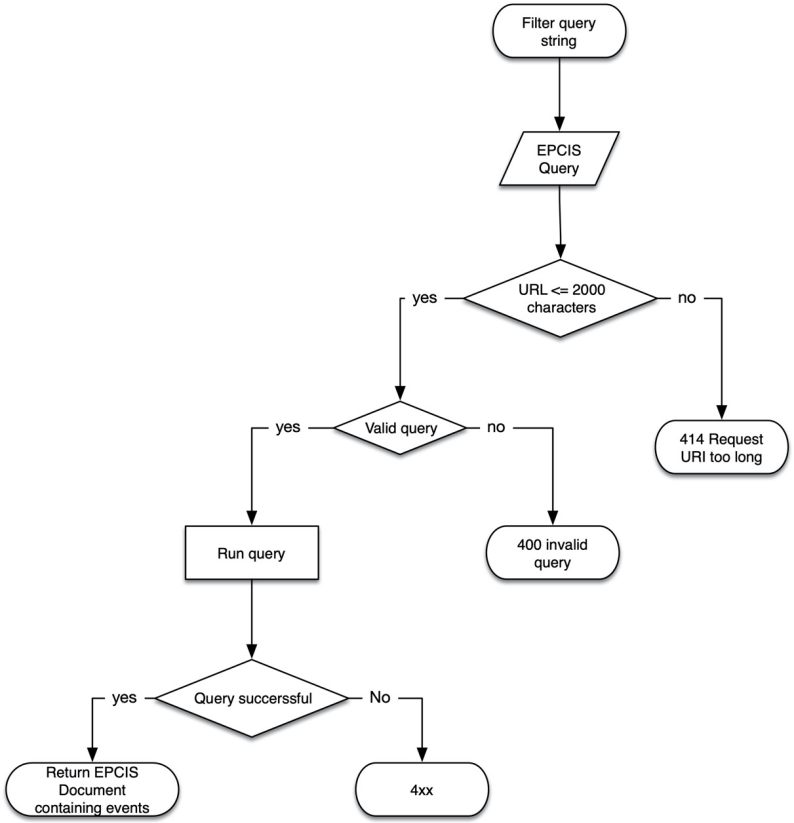
EPCIS 2.0 supports a number of optional top-level resources (see § 12.7.4). By definition, top-level resources SHALL NOT be nested. For example, `/eventTypes/ObjectEvent/events` and `/readPoints/urn:epc:id:sgln:0614141.00777.0/events` are valid paths. `/readPoints/urn:epc:id:sgln:0614141.00777.0/bizSteps/urn:epcglobal:cbv:bizstep:shipping/events` is an invalid path because top-level resource nesting is currently not supported.

12.7.2 EPCIS events endpoints

Each EPCIS event type URL SHALL be identified by a unique URL, such as https://example.com/eventTypes/ObjectEvent/events/ for the collection of object events. Furthermore, if the `eventID` is populated, then each event SHALL use one of the Event IDs specified in the CBV. The event can then be referenced elsewhere (e.g., within an EDI message or Blockchain transaction).

12.7.3 Event filter control interface

Figure 12-4 EPCIS query as URL query parameters



EPCIS defines a domain-specific **query language**, described in § 8.2, as well as dedicated **endpoints for queries**, described in § 13. That selection describes how to filter events using the EPCIS query language in the URL.

EPCIS query as URL query parameters
`https://example.com/eventTypes/all/events?EQ_bizStep=urn:epcglobal:cbv:bizstep:shipping`

URLs MAY be percent-encoded where appropriate, as defined in [RFC3986]. However, when the characters `/` `?` `=` and `&` are used with their special meanings to delimit the structural components of a URL or URL query string, such characters SHALL NOT be percent encoded.

Accessing collection of EPCIS events using EPCIS query or path values
Please note that some queries can also be expressed without query string parameters as a simpler EPCIS 2.0 REST resource. For example

`https://example.com/eventTypes/all/events?EQ_bizStep=urn:epcglobal:cbv:bizstep:shipping`

is functionally equivalent to

`https://example.com/bizSteps/urn:epcglobal:cbv:bizstep:shipping/events`

Expression of query parameters within the URL query string is much more appropriate for complex queries.

12.7.4 Top-level resources

EPCIS 2.0 REST bindings aim to provide an intuitive interface for a variety of use cases and industries, while keeping the API lean and straightforward. This section discusses the mandatory top-level resources and how Comprehensive Business Vocabulary [CBV] extensions can be used to address explore existing resources and to address EPCIS events. An EPCIS 2.0 server MAY provide additional endpoints to describe the context an EPCIS event.

A server SHALL support the top-level resource `/epcs` and MAY support `/bizLocations`, `/bizSteps`, `/readPoints` and `/dispositions`.

The recommended additional top-level resources are:

Table 12-6 Top-level resources endpoints

Top-level resource	As RESTful resource	Description
Electronic product code	`/epcs`	Returns all electronic product codes.
	`/epcs/{epc}`	Returns resource `events`.
	`/epcs/{epc}/events`	Returns all EPCIS events for the electronic product code.
	`/epcs/{epc}/events/{eventID}`	Returns EPCIS event(s) associated with the given EPC and eventID.
Business location	`/bizLocations`	Returns all business locations.
	`/bizLocations/{bizLocation}`	Returns resource `events`.

	/bizLocations/{bizLocation}/events	Returns all EPCIS events in a given business location.
	/bizLocations/{bizLocation}/events/{eventID}	Returns EPCIS event(s) associated with the given business location and eventID.
Business step	/bizSteps	Returns all business steps.
	/bizSteps/{bizStep}	Returns the relative link events
	/bizSteps/{bizStep}/events	Returns all EPCIS events associated with the business step.
	/bizSteps/{bizStep}/events/{eventID}	Returns EPCIS event(s) associated with the gven business step and eventID.
Read point	/readPoints	Returns all read points.
	/readPoints/{readPoint}	Returns the relative link events
	/readPoints/{readPoint}/events	Returns all EPCIS events associated with the read point.
	/readPoints/{readPoint}/events/{eventID}	Returns EPCIS event(s) associated with the given read point and eventID.
Disposition	/dispositions	Returns all dispositions.
	/dispositions/{disposition}	Returns the relative link events
	/dispositions/{disposition}/events	Returns all EPCIS events with the disposition.
	/dispositions/{disposition}/events/{eventID}	Returns single EPCIS event if it the disposition matches.

NOTE: Query endpoints ending with /events/{eventID} may return more than one event if the eventID is present in subsequent events that include an ErrorDeclaration to indicate that the original event was in error.

By default, when a value of {bizLocation}, {bizStep}, {readPoint} or {disposition} is expressed without a namespace qualifier within an endpoint URL, the value SHALL be considered to be using the GS1 Comprehensive Business Vocabulary (CBV). If a value is not

using the CBV, the URI of the vocabulary SHALL be specified in the `GS1-CBV-Extension` header as a mapping to a Compact URI Expression [CURIE] prefix and the value itself SHALL be expressed as a Compact URI Expression [CURIE]. If `GS1-CBV-Extension` contains more than one vocabulary, each vocabulary SHALL be separated by a ",". The motivation for introducing the `GS1-CBV-Extension` header was to provide for simpler URLs while maintaining the flexibility to use custom vocabularies. If the `GS1-CBV-Extension` header is set and the complete resource address is set, specifying the complete resource address for top-level resources SHALL take precedence over the vocabulary specified in the `GS1-CBV-Extension` header.

Example: `GS1-CBV-Extension` **header omitted**
Path: `/bizSteps/urn:epcglobal:cbv:bizstep:shipping/events` Or Path: `/bizSteps/shipping/events` Expands to `urn:epcglobal:cbv:bizstep:shipping` because no vocabulary was specified in the header.
Example: `"GS1-CBV-Extension" : {"ex1" : "<https://example.com/vendor1#>"}`
Path: `/bizSteps/urn:epcglobal:cbv:bizstep:shipping/events` Expands to `urn:epcglobal:cbv:bizstep:shipping` because the CBV namespace was defined, even though `GS1-CBV-Extension` points to a different vocabulary.
Example: `"GS1-CBV-Extension" : {"ex1": "<https://example.com/vendor1#>"}`
Path: `/bizSteps/ex1:qualityCheck/events` Expands to: `https://example.com/vendor1#qualityCheck` because `ex1` is a prefix defined in `GS1-CBV-Extension`.
Example: `"GS1-CBV-Extension" : "ex1 : <https://example.com/vendor1#, vendor2:<https://example.org/defs#> "`
Path: `/bizSteps/vendor2:QA/events` Expands to: `https://example.org/defs#QA` because `vendor2` is one of two vocabulary prefixes defined in `GS1-CBV-Extension`.

12.8 Query control interface

EPCIS SHALL support named queries through the `/queries` endpoint. The EPCIS query language is defined in § 12.9.

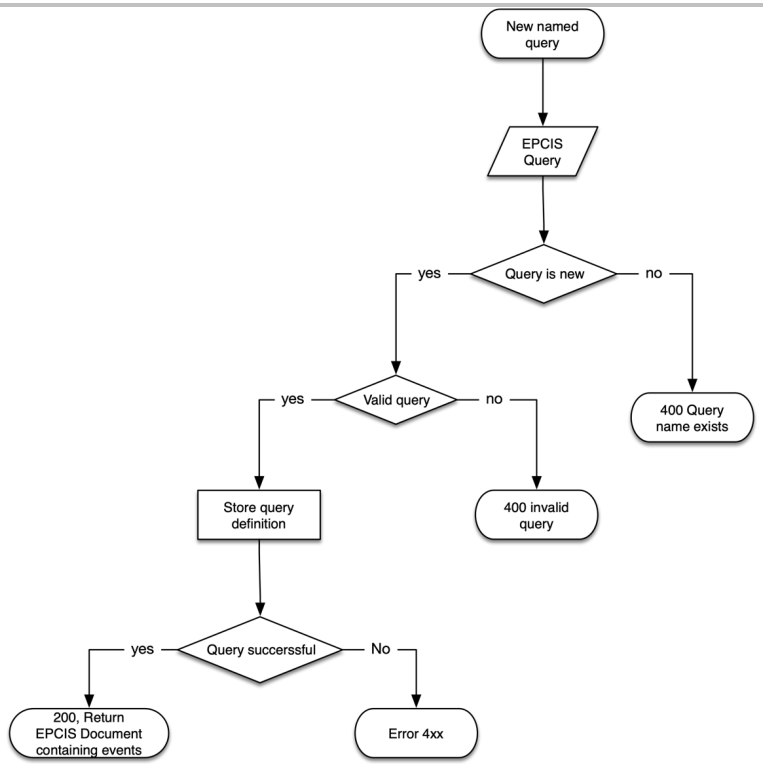


Figure 12-5 Endpoint: Named queries workflow

A named query resembles a stored procedure. It consists of a human-readable name and a query definition. The name SHALL be unique, as it is part of a query's URL. The query SHALL be defined on creation time and transmitted in the request body.

Table 12-7 EPCIS query endpoints

Endpoint	**Verb**		
	GET	POST	DELETE
/queries	Returns all named queries.		
/queries/{queryName}	Returns the named query	Creates and executes a named EPCIS events or	Removes a named query and closes open

		EPCIS master data query and returns results with the option to use pagination if needed.	WebSocket connections to clients.
`/queries/{queryName}/events`	Returns all EPCIS events that match the query or creates a new Websocket subscriptions.		
`/queries/{queryName}/subscriptions`	Returns active Webhook subscriptions with the option to use pagination if needed.	Creates a new query subscription	
`/queries/{queryName}/subscriptions/{subscriptionID}`	Information about a Webhook subscription		Unsubscribes the client.
`/queries/{queryName}/masterData`	Returns the EPCIS vocabulary elements.		
`/queries/SimpleMasterDataQuery`	Returns the string `masterData`	Executes anonymous EPCIS master data query and returns results with the option to use pagination if needed.	
`/queries/SimpleMasterDataQuery/masterData`	Returns EPCIS vocabulary items that did not fit inside the response of the `POST` of the anonymous query.		
`/queries/SimpleEventQuery`	Returns the string `events`	Executes anonymous EPCIS events query and returns results with the option to use pagination if needed.	

`/queries/SimpleEventQuery/events`	Returns EPCIS events that did not fit inside the response of the `POST` of the anonymous query.

12.8.1 Creating and using named queries

A named query creates a virtual collection of events (i.e., a view). It has a human-readable name and a query defined using the EPCIS Query Language. The result set of an EPCIS events query has the following URL pattern: `/queries/{queryName}/events`. EPCIS queries also support master data. The result set of a named query for master data is `/queries/{queryName}/events`.

A named query is created by specifying the query name, the type of query (EPCIS events or master data) and the query body. Creating a query also returns the result set. The client can use the Link header to obtain the remaining resources using pagination. If the query is for EPCIS events, the query results endpoint is `events`. If the query is for master data, the results endpoint is `masterData`. The client can also fetch all events that match the query description using these endpoints.

Once a query is created, clients can always retrieve the current set of resources that match the query description.

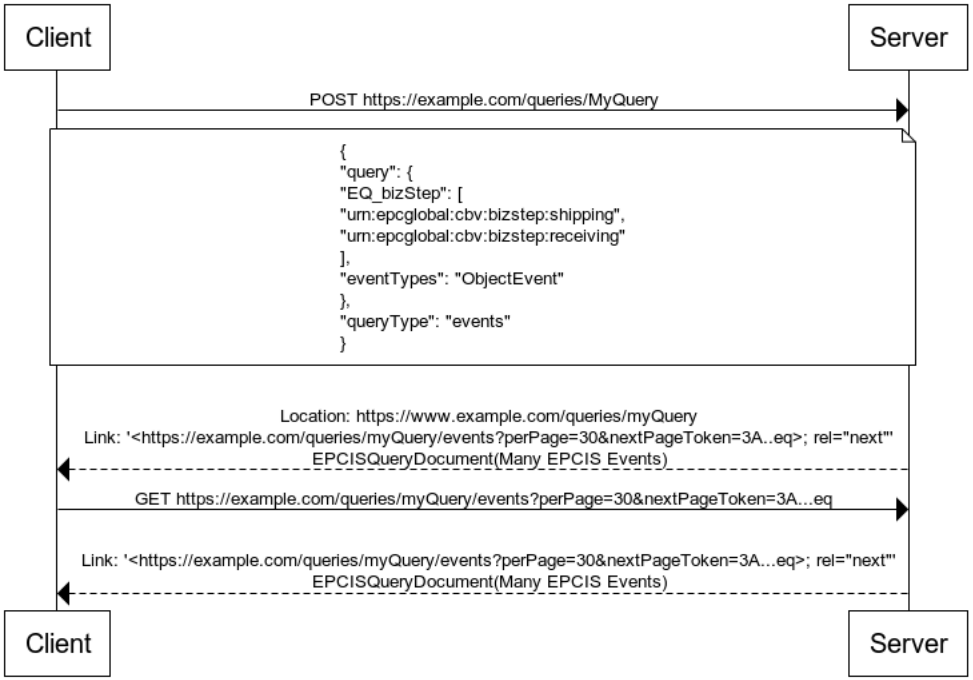


Figure 12-6 Client creates a named query for EPCIS events and uses pagination to retrieve all EPCIS events.

12.8.2 Deleting named queries

A named query can be deleted by calling the `DELETE` method on its URL. Deleting a named query also deletes all query subscriptions and disconnects all clients that are subscribed to query subscriptions.

12.8.3 Subscribing to named queries

A named EPCIS events query SHALL support subscription using HTTP callbacks (aka Webhooks) and MAY support subscription using WebSockets. The server SHALL guarantee that clients only receive events they are authorised to receive. When connecting to a query subscription, the client SHALL have the option to specify from when onwards the query applies with the initial record time (`initialRecordTime` parameter). If the initial record time is omitted, it defaults to the time at which the subscription was created.

By default, a client SHALL only receive a query result when the query result is not empty. The client can overwrite this behaviour by setting the report if empty (`reportIfEmpty` parameter) to `true`. This means that the client will receive an empty query result set when the query

does not match any events. The server MAY implement the minimum record time (`minRecordTime` parameter), which specifies the lower bound (in time) of EPCIS events that are contained in the result set. If the server implements support for minimum record time it the value should be set to the most recent execution time of the query subscription.

To ensure that clients receive EPCIS events when the connection between client and server is temporarily lost, the server MAY provide a quality of service that guarantees a message is delivered at least one time to the receiver. If the server provides quality of service and based on the minimum record time, the server SHOULD return all EPCIS events starting from the minimum record time.

If the subscription contains an error, the server responds with HTTP status code `400` and indicate the error as `SubscriptionControlsException`.

Table 12-8 Query endpoint usage overview

How to...	Webhooks (HTTP Callbacks) - SHALL	Websockets - MAY
Create a new query	`POST /queries/{queryName}`	`POST /queries/{queryName}`
Fetch events synchronously (via HTTP)	`GET /queries/{queryName}/events`	`GET /queries/{queryName}/events`
Subscribe to query	`POST /queries/{queryName}/subscriptions callbackUrl=client.com/queryCallback`	1. `GET /queries/{queryName}/events` 2. Switch to Websocket protocol
Configure the subscription	As properties subscription request payload.	As URL query string parameters in the subscription request.
Notify subscriber even if the query result is empty	`reportIfEmpty` is true in the subscription request payload.	Set `reportIfEmpty` to true URL query string parameter in the subscription request.
Overwrite the initial record time time for a query subscription	Assign a timestamp to the `initialRecordTime` property in the subscription request payload.	Assign a timestamp to the `initialRecordTime` URL query string parameter in the subscription request.
Track the minimum record time for quality of service	Server maintains the timestamp of the most recent client notification in the `minRecordTime` property of the subscription resource.	Server maintains the timestamp of the most recent client notification in `the GS1-Query-Min-Record-Time` header.
Secure the subscription	Client generates a secret password that is included in the `secret` property in the subscription request payload.	No additional step is needed.
Publish query results (Server to client)	`POST https://client.com/queryCallback EPCISDocument...`	`WebSocket.send(EPCISDocument)`
Unsubscribe	`DELETE /queries/{queryName}/subscriptions/s123`	`WebSocket.close()`

12.8.3.1 Scheduled queries

Scheduled queries are cron-like [CRON] queries that are executed in predefined time intervals. A scheduled query SHOULD push a list of events that match the query description to all subscribed clients. A scheduled query is not defined by the query name but by the schedule, which is specified in the URL query string. The REST API schedule uses the same query schedule parameters and meanings as § 8.2.5.3.

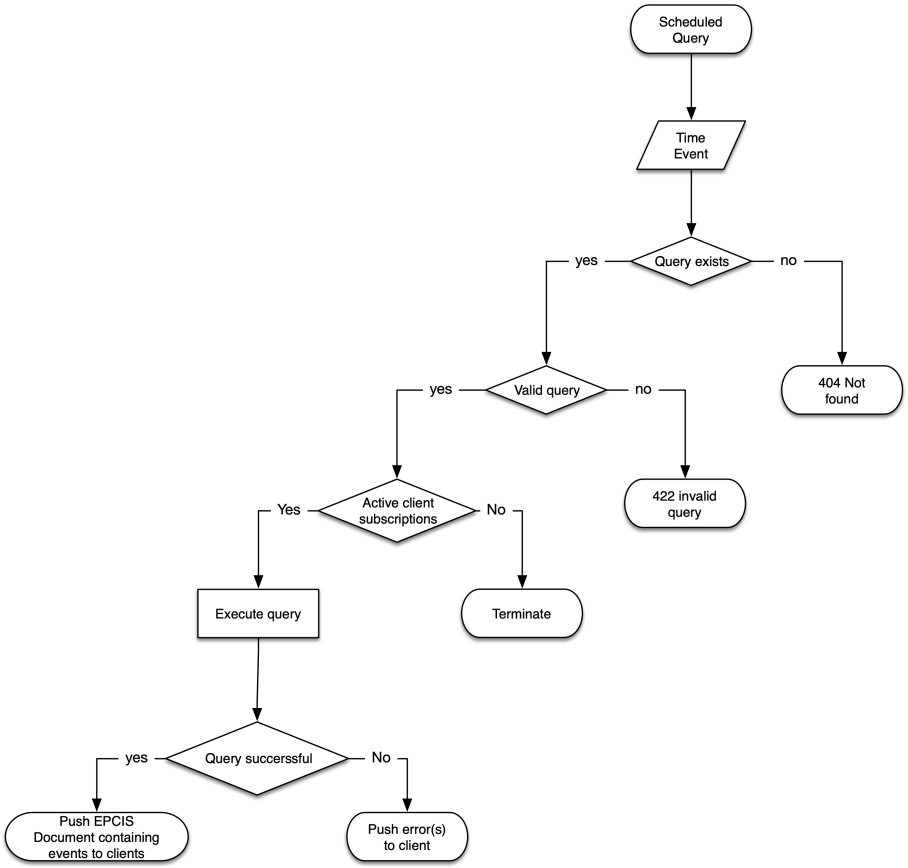


Figure 12-7 Scheduled query workflow

12.8.3.2 Streaming queries

If no query schedule is specified, the client must explicitly set `stream` to `true`. This restriction is to prevent clients from accidentally subscribing to EPCIS event streams. Whenever a captured EPCIS event matches the query criteria, the client is notified. The query SHALL only execute a new event matches the query in full. Upon execution, all subscribed clients SHOULD receive the list of events.

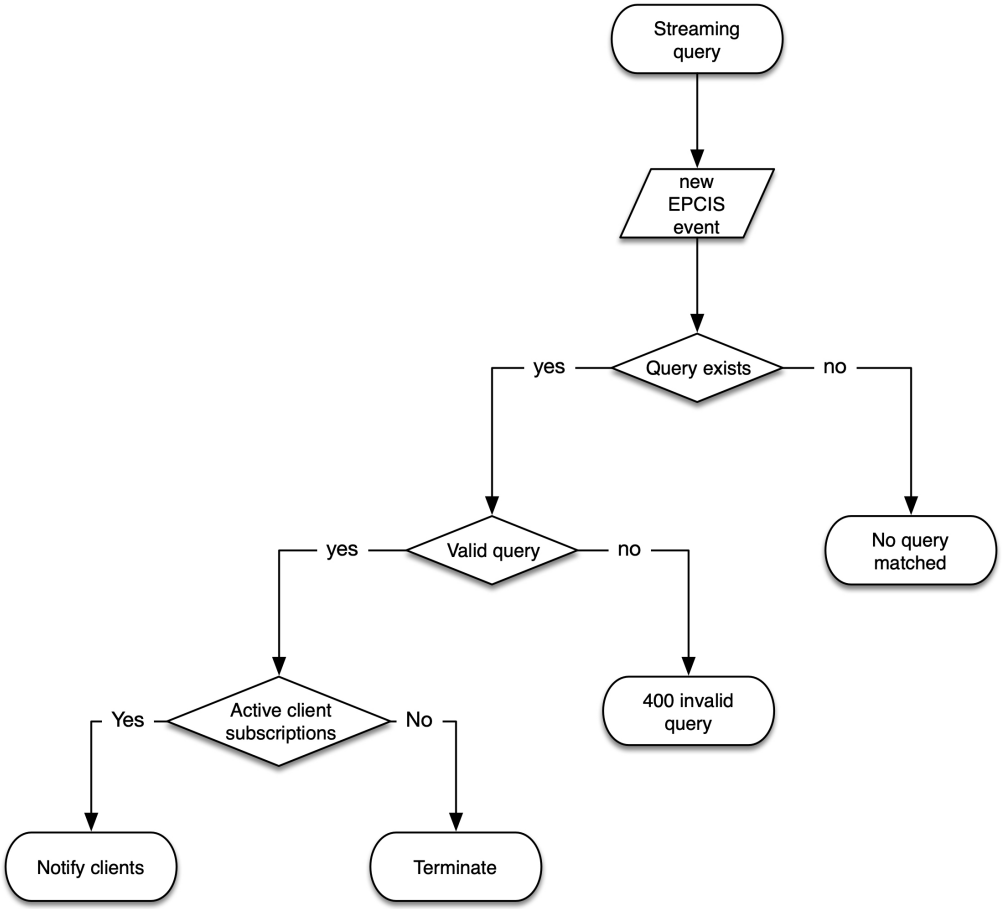


Figure 12-8 Event streaming query workflow

Scheduled query conditions and `stream` flag SHALL be mutually exclusive. A query SHALL execute only if the entire query schedule matches.

12.8.3.2.1 Webhook (HTTP Callbacks) for query subscription

Creating query subscription using Webhooks requires the client to provide a single endpoint to which the server will send events and a subscription secret that the client needs to authenticate itself when sending events. The subscription secret must be generated by the client. When the client subscribes to a query, it must either set `stream` to `true`, to be notified whenever a new EPCIS event matches the query, or the client must define a query schedule. If these are missing the query subscription is invalid because the server won't know when to notify a client.

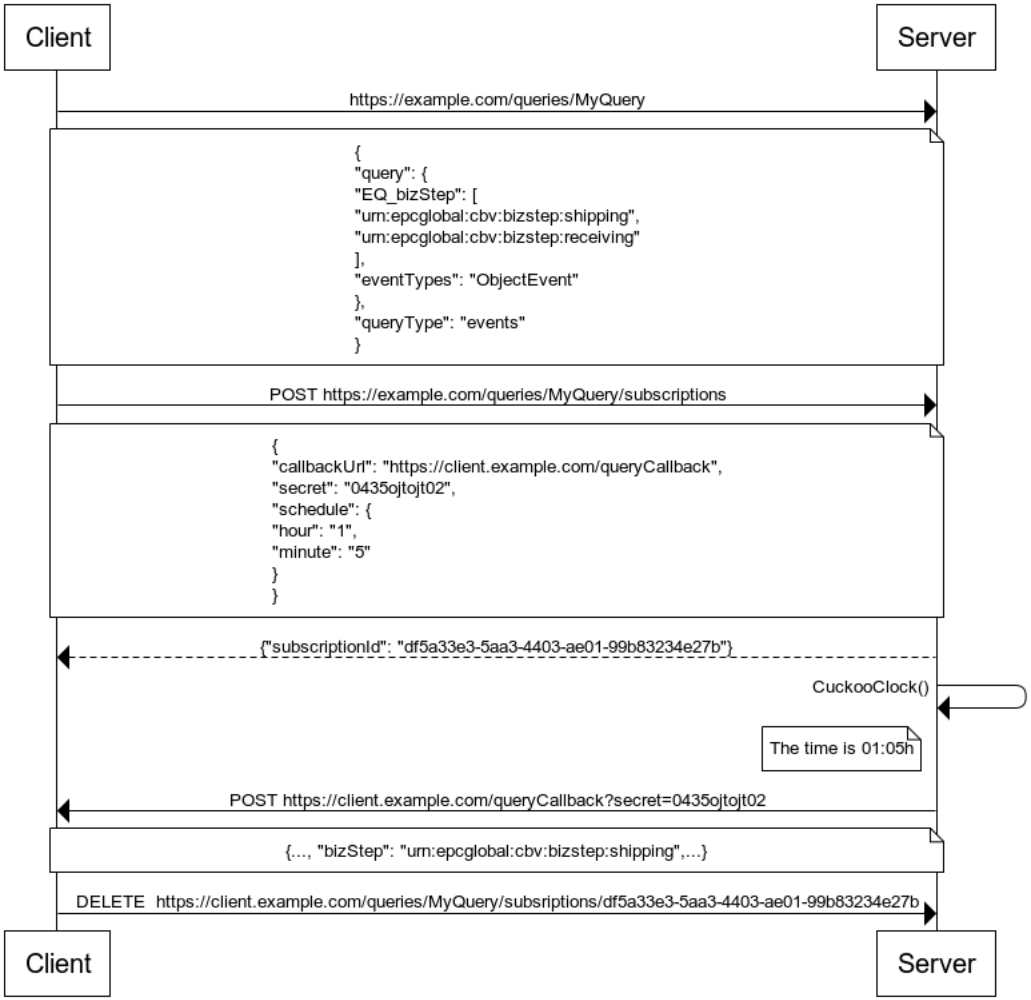


Figure 12-9 Query subscription with Webhook (HTTP Callback)

12.8.3.3 WebSocket for query subscription

In addition to Webhooks, EPCIS 2.0 adds the **WebSocket** protocol, which maintains a persistent connection without requiring the client to run a Web server. An EPCIS server MAY support query subscription based on the WebSocket protocol for scheduled queries and for

streaming queries. Websocket support is provided through the `/queries/{namedQueries/events` endpoints. Unlike Webhook subscriptions, Websocket query subscription parameters are specified as URL query string parameters.

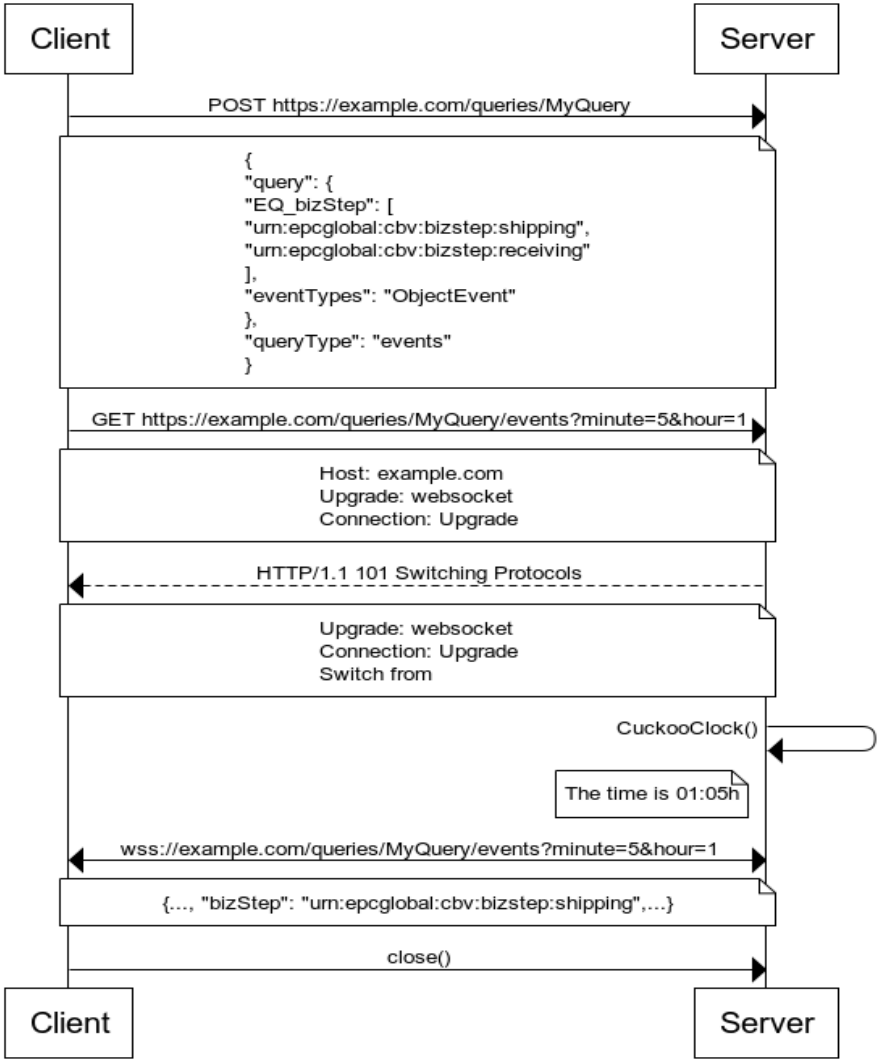


Figure 12-10 Query subscription with a WebSocket

12.8.1 Anonymous Queries

Anonymous queries provide support for large EPCIS queries that are short lived. Unlike named queries, anonymous queries are not persisted. Anonymous queries work by perfomring a POST on `/queries/SimpleEventQuery` for EPCIS events and `/queries/SimpleMasterDataQuery` for EPCIS master data. Anonymous queries follow the same pattern as named queries. First, the server responds to the anonymous queries by returning resources that match. Second, the client can use the URL in the `Link` header to fetch remaining resources. Unlike named queries, query subscription SHALL not supported because anonymous queries are designed to be short lived.

12.9 EPCIS query language

EPCIS queries are constructed using an EPCIS specific query language. The simplest query consists of an operand name followed by an underscore "_" and an EPCIS resource property name (e.g., `GE_eventTime("2005-07-11T11:30:47.0Z")`). A compound query can specify conditions for more than one property. Implicitly, a logical "AND" conjunction connects the clauses in a compound query, e.g., `GE_eventTime("2005-07-11T11:30:47.0Z"), eventTypes('ObjectEvent')`.

EPCIS queries can be expressed as query string parameters or as an EPCIS JSON query document. An EPCIS 2.0 server SHALL support the URL query string parameters syntax and SHOULD support the JSON EPCIS query document syntax. The EPCIS JSON query document syntax is recommended when the URL length exceeds 2000 characters.

12.9.1 EPCIS query in the URL

The URL length SHALL be limited to 2000 characters. This limitation is to guarantee support for most browsers and Web servers. An EPCIS query in the URL SHALL be percent-encoded (see [RFC3986]) and inserted in the URL following "`?`". The "`=`" infix operator SHALL have the query parameter on the left and value on the right. If the EPCIS query parameter supports an array of values, each value SHALL be separated by ",". Each clause within the URL query string SHALL be connected by an "`&`".

Example EPCIS query as escaped query parameters

```
https://example.com/eventTypes/ObjectEvents/events?
EQ_bizStep=urn%3Aepcglobal%3Acbv%3Abizstep%3Ashipping%2Curn%3Aepcglobal%3Acbv%3Abizstep%3Adecommissioning&GE_eventTi
me=2015-03-15T00%3A00%3A00.000-04%3A00
```

12.9.2 Query document syntax

An EPCIS 2.0 query document SHALL be serialised as a JSON array. Each EPCIS query parameter SHALL be an item in the previously mentioned array, serialised as a JSON object/dictionary. Each item SHALL be a valid query, with the object/dictionary key containing the EPCIS query parameter and the value indicating either a single matching value or an array/list of values, for which one value is required to match.

`POST /queries/{queryName}` and `POST /queries/SimpleEventQuery` endpoints SHALL accept a request body in application/ld+json (JSON-LD) or application/json (JSON) format . When the request body is in application/ld+json (JSON-LD) format, it MAY contain an `@context` field to specify namespace information for user extension query parameters. These endpoints also support the `GS1-EPCIS-Extensions` request header, which serves the same purpose to allow users to provide namespace information as comma-separated values. If the user provides data in both request header `GS1-EPCIS-Extensions` and field `@context` then the combined effect of both MUST be taken into final consideration. In case of conflicts in which a CURIE prefix is mapped to more than one namespace or URI stem or a JSON key is mapped to more than one URI, the client must be notified with HTTP status code 400 with the reason mentioning the conflicting mappings of keys or namespace prefixes.

GS1 EPCIS 2.0 defines a standard context resource for the JSON-LD format of EPCIS data and any user-defined mappings SHALL NOT conflict with those.

Example:

Example EPCIS 2.0 query body

```
[
  {
    "eventTypes": "ObjectEvent"
  },
  {
    "EQ_bizStep": [
      "urn:epcglobal:cbv:bizstep:shipping",
      "urn:epcglobal:cbv:bizstep:decommissioning"
    ]
  },
  {
    "GE_eventTime": "2015-03-15T00:00:00.000-04:00"
  }
]
```

12.10 Backward Compatibility of REST bindings with EPCIS 1.2

The EPCIS 2.0 REST bindings maintains backward semantic compatibility with EPCIS 1.2. This means that legacy EPCIS repositories and clients will be able to seamlessly interface with EPCIS 2.0 REST bindings through a proxy or adapter which maps between EPCIS 1.2 SOAP endpoints to EPCIS 2.0 REST endpoints. In addition to defining a backwards compatible EPCIS RESTful API, EPCIS 2.0 extends EPCIS 1.2 with characteristics that can be expected from a RESTful webservice.

12.11 EPCIS Error Conditions and HTTP Status Code Mapping

EPCIS 1.2 defines a number of error conditions and their underlying causes that an EPCIS server SHALL return [EPCIS1.2]. This section maps error conditions as defined in [EPCIS1.2] to HTTP status codes. Error responses SHALL be formatted using the "Problem Details for HTTP APIs" [RFC7807] format.

The following table HTTP status codes to EPCIS exceptions. An EPCIS 2.0 server SHALL return HTTP status codes and SHALL include the exceptions in the body and MAY provide additional information regarding the exception.

Table 12-9 EPCIS exceptions and HTTP status codes

EPCIS Exceptions	HTTP Status Codes
epcisException:ValidationException	400
epcisException:QueryValidationException	400
epcisException:QueryParameterException	400
epcisException:SubscriptionControlsException	400
epcisException:SecurityException	403, 401
epcisException:NoSuchNameException	404
epcisException:CaptureLimitExceededException	413
epcisException:QueryTooLargeException	413
epcisException:QueryTooComplexException	413
epcisException:URITooLongException	414
epcisException:ImplementationException	501, 500

The `type` field of a problem response body expects a URI. The server MAY use the Compact URI Expression with `epcisExceptions` as namespace. For example, if the server encounters a SecurityException, it will respond with:

Example: Client is lacks valid authentication to access /events

```
{
  "type": "epcisExceptions:SecurityException",
```

```
  "title": "Access denied",
  "status": 401,
  "detail": "User is not permitted to access /events"
}
```

2901

12.12 Conformance Statement for EPCIS 2.0 Servers

1. An EPCIS server SHALL support the top-level resources `/epcs` and `/eventTypes` and SHOULD support `/bizLocations`, `/bizSteps`, `/readPoints` and `/dispositions` and these resource SHALL comply with the following pattern:
   `/{top-level resource}s/{optional vocabulary prefix}:{resource identifier}/events`
2. An EPCIS server SHALL support named queries through the `/queries` endpoint
3. A server SHALL implement the virtual event type "`all`" as an additional collection which contains all EPCIS events in the server's repository regardless of their types.
4. Each endpoint SHALL support HTTP content negotiation [HTTPSemanticsContent] for at least JSON [JSON] and JSON-LD [JSONLD. 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`.
5. For all endpoints, a server SHALL support `GS1-CBV-version`, `GS1-CBV-extensions`, `GS1-EPCIS-version`, `GS1-EPCIS-min` and `GS1-EPCIS-max`.
6. 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 octets.
7. A server SHALL support `GS1-Capture-Error-Behaviour` to declare the error behaviour of the capture interface.
8. If `GS1-Capture-Error-Behaviour` is `rollback`, the server SHALL guarantee that either all events are captured, or all events are rejected. If `GS1-Capture-Error-Behaviour` is `proceed`, the server SHALL try to capture as many EPCIS events as possible.
9. The server SHALL support the `OPTIONS` method for each endpoint and SHALL provide default header values for each custom EPCIS header it supports.
10. EPCIS clients SHALL authenticate themselves for every API call.
11. An EPCIS repository SHALL implement pagination, to return events in manageable chunks.
12. If the payload is serialised in XML or JSON/JSON-LD, events SHALL conform to the XML or JSON/JSON-LD event specifications, respectively, specified in EPCIS [EPCIS2.0].
13. The query SHALL only execute if the entire query schedule matches or if it is a streaming query.
14. If the client sends a payload that exceeds the number of events permitted according to `GS1-EPCIS-Capture-Limit` or the content length exceeds `GS1-EPCIS-Capture-File-Size-Limit`, the server SHALL respond with `413 Payload Too Large` and include the most restrictive capture limit constraint.
15. When calling the `GET` method on the `/eventTypes` endpoint, it SHALL return a list of all EPCIS event types that are present in that repository.

16. By default, when a value of {epc}, {bizLocation}, {bizStep}, {readPoint} or {disposition} is expressed without a namespace qualifier within an endpoint URL, the value SHALL be considered to be using the GS1 Comprehensive Business Vocabulary (CBV). If a value is not using the CBV, the URI of the vocabulary SHALL be specified in the `GS1-CBV-Extension` header as a mapping to a Compact URI Expression [CURIE] prefix and the value itself SHALL be expressed as a Compact URI Expression [CURIE]. If `GS1-CBV-Extension` contains more than one vocabulary, each vocabulary SHALL be separated by a ",". The motivation for introducing the `GS1-CBV-Extension` header was to provide for simpler URLs while maintaining the flexibility to use custom vocabularies. If the `GS1-CBV-Extension` header is set and the complete resource address is set, specifying the complete resource address for top-level resources SHALL take precedence over the vocabulary specified in the `GS1-CBV-Extension` header.
17. EPCIS SHALL support named queries through the `/queries` endpoint.
18. A named EPCIS events query SHALL support subscription using Webhooks (HTTP Callbacks).
19. Scheduled query conditions and streamed query conditions SHALL be mutually exclusive.
20. An EPCIS 2.0 server SHALL support the URL query string parameters syntax and SHOULD support the JSON EPCIS query document syntax.
21. EPCIS 1.2 defines a number of error conditions and their underlying causes that an EPCIS server SHALL return [EPCIS1.2]. Error responses SHALL be formatted using the "Problem Details for HTTP APIs" [RFC7807] standard.
22. An EPCIS 2.0 server SHALL return HTTP status codes and SHALL include the exceptions in the body and MAY provide additional information regarding the exception.

13 Bindings for core query operations module

This section defines bindings for the Core Query Operations Module, as follows:

Interface	Binding	Document section
Query Control Interface	SOAP over HTTP (WSDL)	§ 13.2
	XML over AS2	§ 13.3
Query Callback Interface	XML over HTTP	§ 13.4.2
	XML over HTTP+TLS (HTTPS)	§ 13.4.3
	XML over AS2	§ 13.4.4

All of these bindings share a common XML syntax, specified in § 13.1. The XML schema has the following ingredients:

- XML elements for the argument and return signature of each method in the Query Control Interface as defined in § 8.2.5
- XML types for each of the datatypes used in those argument and return signatures
- XML elements for each of the exceptions defined in § 8.2.6
- XML elements for the Query Callback Interface as defined in § 8.2.8. (These are actually just a subset of the previous three bullets.)
- An `EPCISQueryDocument` element, which is used as an "envelope" by bindings whose underlying technology does not provide its own envelope or header mechanism (specifically, all bindings except for the SOAP binding). The AS2 binding uses this to provide a header to match requests and responses. The `EPCISQueryDocument` element shares the `EPCISHeader` type defined in § 9.5. Each binding specifies its own rules for using this header, if applicable.

13.1 XML schema for core query operations module

The XML schema `EPCglobal-epcis-query` defines XML representations of data types, requests, responses, and exceptions used by the EPCIS Query Control Interface and EPCIS Query Callback Interface in the Core Query Operations Module. This schema is incorporated by reference into all of the bindings for these two interfaces specified in the remainder of this § 12. This schema SHOULD be used by any new binding of any interface within the Core Query Operations Module that uses XML as the underlying message format.

The `QueryParam` type defined in the schema is used to represent a query parameter as used by the `poll` and `subscribe` methods of the query interface defined in § 8.2.5. A query parameter consists of a name and a value. The XML schema specifies `xsd:anyType` for the value, so that a parameter value of any type can be represented. When creating a document instance, the actual value SHALL conform to a type appropriate for the query parameter, as defined in the following table:

Parameter type	XML type for `value` element
AnyURI	`xsd:anyURI`

Parameter type	XML type for `value` element
Boolean	`xsd:boolean`
Date	`xsd:date`
Decimal	`xsd:decimal`
Double	`xsd:double`
HeBinary	`xsd:hexBinary`
Int	`xsd:integer`
Float	`xsd:float`
DateTimeStamp	`xsd:dateTimeStamp`
String	`xsd:string`
List of String	`epcisq:ArrayOfString`
Void	`epcisq:VoidHolder`

In particular, the table above SHALL be used to map the parameter types specified for the predefined queries of § *8.2.7* into the corresponding XML types.

Each `<value>` element specifying a query parameter value in an instance document MAY include an `xsi:type` attribute as specified in [XSD1]. The following rules specify how query parameter values are processed:

- When a `<value>` element does not include an `xsi:type` attribute, the `subscribe` or `poll` method of the Query Control Interface SHALL raise a `QueryParameterException` if the specified value is not valid syntax for the type required by the query parameter.
- When a `<value>` element does include an `xsi:type` attribute, the following rules apply:
  - If the body of the `<value>` element is not valid syntax for the type specified by the `xsi:type` attribute, the `EPCISQueryDocument` or SOAP request MAY be rejected by the implementation's XML parser.
  - If the value of the `xsi:type` attribute is not the correct type for that query parameter as specified in the second column of the table above, the `subscribe` or `poll` method of the Query Control Interface MAY raise a `QueryParameterException`, even if the body of the `<value>` element is valid syntax for the type required by the query parameter.
  - If the body of the `<value>` element is not valid syntax for the type required by the query parameter, the `subscribe` or `poll` method of the Query Control Interface SHALL raise a `QueryParameterException` unless the `EPCISQueryDocument` or SOAP request was rejected by the implementation's XML parser according to the rule above.

This schema imports additional schemas as shown in the following table:

Namespace	Location reference	Source
urn:epcglobal:xsd:1	EPCglobal.xsd	§ 9.3
http://www.unece.org/cefact/namespaces/StandardBusinessDocumentHeader	StandardBusinessDocumentHeader.xsd	UN/CEFACT web site; see § 9.2
urn:epcglobal:epcis:xsd:1	EPCglobal-epcis-1_0.xsd	§ 9.5

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 of type `xsd:anyURI` or `xsd:string` that specifies `minOccurs="0"`, 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 schema for the Core Query Operations Module can be found in the external artefact, " EPCglobal-epcis-query.xsd":

13.2 SOAP/HTTP binding for the query control interface

The external Web Service Description Language (WSDL) 1.1 [WSDL1.1] specification defining the standard SOAP/HTTP binding of the EPCIS Query Control Interface. An EPCIS implementation MAY provide a SOAP/HTTP binding of the EPCIS Query Control Interface; if a SOAP/HTTP binding is provided, it SHALL conform to the following WSDL. This SOAP/HTTP binding is compliant with the WS-I Basic Profile Version 1.0 [WSI]. This binding builds upon the schema defined in § 13.1.

If an EPCIS implementation providing the SOAP binding receives an input that is syntactically invalid according to this WSDL, the implementation SHALL indicate this in one of the two following ways: the implementation MAY raise a `ValidationException`, or it MAY raise a more generic exception provided by the SOAP processor being used.

13.3 AS2 Binding for the query control interface

This section defines a binding of the EPCIS Query Control Interface to AS2 [RFC4130]. An EPCIS implementation MAY provide an AS2 binding of the EPCIS Query Control Interface; if an AS2 binding is provided it SHALL conform to the provisions of this section. For the purposes of this binding, a "query client" is an EPCIS Accessing Application that wishes to issue EPCIS query operations as defined in § 8.2.5, and a "query server" is an EPCIS Repository or other system that carries out such operations on behalf of the query client.

A query server SHALL provide an HTTP URL through which it receives messages from a query client in accordance with [RFC4130]. A message sent by a query client to a query server SHALL be an XML document whose root element conforms to the `EPCISQueryDocument` element as defined by the schema in § 13.1. The element immediately nested within the `EPCISBody` element SHALL be one of the elements corresponding to an EPCIS Query Control Interface method request (i.e., one of `Subscribe`, `Unsubscribe`, `Poll`, etc.). The permitted elements are listed in the table below. If the message sent by the query client fails to conform to the above requirements, the query server SHALL respond with a `ValidationException` (that is, return an `EPCISQueryDocument` instance where the element immediately nested within the `EPCISBody` is a `ValidationException`).

The query client SHALL provide an HTTP URL that the query server will use to deliver a response message. This URL is typically exchanged out of band, as part of setting up a bilateral trading partner agreement (see [RFC4130] § 5.1).

Both the query client and query server SHALL comply with the Requirements and SHOULD comply with the Recommendations listed in the GS1 document "EDIINT AS1 and AS2 Transport Communications Guidelines" [EDICG]. For reference, the relevant portions of this document are reproduced below.

The query client SHALL include the Standard Business Document Header within the `EPCISHeader` element. The query client SHALL include within the Standard Business Document Header a unique identifier as the value of the `InstanceIdentifier` element. The query client MAY include other elements within the Standard Business Document Header as provided by the schema. The instance identifier provided by the query client SHOULD be unique with respect to all other messages for which the query client has not yet received a corresponding response. As described below, the instance identifier is copied into the response message, to assist the client in correlating responses with requests.

A query server SHALL respond to each message sent by a query client by delivering a response message to the URL provided by the query client, in accordance with [RFC4130]. A response message sent by a query server SHALL be an XML document whose root element conforms to the `EPCISQueryDocument` element as defined by the schema in § 13.1. The element immediately nested within the `EPCISBody` element SHALL be one of the elements shown in the following table, according to the element that was provided in the corresponding request:

Request element	Permitted return elements

`GetQueryNames`	`GetQueryNamesResult`
	`SecurityException`
	`ValidationException`
	`ImplementationException`

`Subscribe`	`SubscribeResult`
	`NoSuchNameException`
	`InvalidURIException`
	`DuplicateSubscriptionException`
	`QueryParameterException`
	`QueryTooComplexException`
	`SubscriptionControlsException`
	`SubscribeNotPermittedException`
	`SecurityException`
	`ValidationException`
	`ImplementationException`

`Unsubscribe`	`UnsubscribeResult`
	`NoSuchSubscriptionException`
	`SecurityException`
	`ValidationException`
	`ImplementationException`

Request element	Permitted return elements
GetSubscriptionIDs	GetSubscriptionIDsResult NoSuchNameException SecurityException ValidationException ImplementationException
Poll	QueryResults QueryParameterException QueryTooLargeException QueryTooComplexException NoSuchNameException SecurityException ValidationException ImplementationException
GetStandardVersion	GetStandardVersionResult SecurityException ValidationException ImplementationException
GetVendorVersion	GetVendorVersionResult SecurityException ValidationException ImplementationException

The query server SHALL include the Standard Business Document Header within the EPCISHeader element. The query server SHALL include within the Standard Business Document Header the `BusinessScope` element containing a `Scope` element containing a `CorrelationInformation` element containing a `RequestingDocumentInstanceIdentifier` element; the value of the latter element SHALL be the value of the `InstanceIdentifier` element from the Standard Business Document Header of the corresponding request. Within the `Scope` element, the `Type` subelement SHALL be set to `EPCISQuery`, and the `InstanceIdentifier` element SHALL be set to `EPCIS`. The query server MAY include other elements within the Standard Business Document Header as provided by the schema.

13.3.1 GS1 AS2 guidelines (Non-Normative)

As stated above, the query client and query server SHALL comply with the Requirements and SHOULD comply with the Recommendations listed in the GS1 document "EDIINT AS1 and AS2 Transport Communications Guidelines" [EDICG] For reference, the relevant portions of this document are reproduced below. This extract is marked non-normative; in the case of conflict between [EDICG] and what is written below, [EDICG] shall prevail.

Digital Certificate Requirements

Requirement 1

Payload data SHALL be encrypted and digitally signed using the S/MIME specification (see RFC 3851).

Requirement 2

The length of the one-time session (symmetric) key SHALL be 128 bits or greater.

Requirement 3

The length of the Public/Private Encryption key SHALL be 1024 bits or greater.

Requirement 4

The length of the Public/Private Signature key SHALL be 1024 bits or greater.

Requirement 5

The Signature Hash algorithm used SHALL be SHA1.

Configuration Requirement

Requirement 6

Digitally signed receipts (Signed Message Disposition Notifications (MDNs)) SHALL be requested by the Sender of Message.

Recommendations

Recommendation 1 – MDN Request Option

Either Asynchronous or Synchronous MDNs MAY be used with EDIINT AS2. There are potential issues with both synchronous and asynchronous MDNs, and Trading Partners need to jointly determine which option is best based on their operational environments and message characteristics.

Recommendation 2 – MDN Delivery

Recipients SHOULD transmit the MDN as soon as technically possible to ensure that the message sender recognises that the message has been received and processed by the receiving EDIINT software in a timely fashion. This applies equally to AS1 and AS2 as well as Asynchronous and Synchronous MDN requests.

Recommendation 3 – Delivery Retry with Asynchronous MDNs Requested

When a message has been successfully sent, but an asynchronous MDN has not been received in a timely manner, the Sender of Message SHOULD wait a configurable amount of time and then automatically resend the original message with the same content and the same Message-ID value as the initial message. The period of time to wait for a MDN and then automatically resend the original message is based on business and technical needs, but generally SHOULD be not be less than one hour. There SHOULD be no more than two automatic resends of a message before personally contacting a technical support contact at the Receiver of Message site.

Recommendation 4 – Delivery Retry for AS2

Delivery retry SHOULD take place when any HTTP response other than "200 OK" or "202 Accepted" is received (for example, 401, 500, 502, 503, timeout, etc). This occurrence indicates that the actual transfer of data was not successful. A delivery retry of a message SHALL have the same content and the same Message-ID value as the initial message. Retries SHOULD occur on a configurable schedule. Retrying SHALL cease when a message is successfully sent (which is indicated by receiving a HTTP 200 range status code, e.g., "200 OK" or "202 Accepted"), or SHOULD cease when a retry limit is exceeded.

Recommendation 5 – Message Resubmission

If neither automated Delivery Retry nor automated Delivery Resend are successful, the Sender of Message MAY elect to resubmit the payload data in a new message at a later time. The Receiver of Message MAY also request message resubmission if a message was lost subsequent to a successful receive. If the message is resubmitted a new Message-ID MUST be used. Resubmission is normally a manual compensation.

Recommendation 6 – HTTP vs. HTTP/S (SSL)

For EDIINT AS2, the transport protocol HTTP SHOULD be used. However, if there is a need to secure the AS2-To and the AS2-From addresses and other AS2 header information, HTTPS MAY be used in addition to the payload encryption provided by AS2. The encryption provided by HTTPS secures only the point to point communications channel directly between the client and the server.

Recommendation 7 – AS2 Header

For EDIINT AS2, the values used in the AS2-From and AS2-To fields in the header SHOULD be GS1 Global Location Numbers (GLNs).

Recommendation 8 - SMTP

[not applicable]

Recommendation 9 - Compression

EDIINT compression MAY be used as an option, especially if message sizes are larger than 1MB. Although current versions of EDIINT software handle compression automatically, this SHOULD be bilaterally agreed between the sender and the receiver.

Recommendation 10 – Digital Certificate Characteristics

Digital certificates MAY either be from a trusted third party or self signed if bilaterally agreed between trading partners. If certificates from a third party are used, the trust level SHOULD be at a minimum what is termed 'Class 2' which ensures that validation of the individual and the organisation has been done.

Recommendation 11 – Common Digital Certificate for Encryption & Signature

A single digital certificate MAY be used for both encryption and signatures, however if business processes dictate, two separate certificates MAY be used. Although current versions of EDIINT software handle two certificates automatically, this SHOULD be bilaterally agreed between the sender and the receiver.

Recommendation 12 – Digital Certificate Validity Period

The minimum validity period for a certificate SHOULD be 1 year. The maximum validity period SHOULD be 5 years.

Recommendation 13 – Digital Certificate – Automated Exchange

The method for certificate exchange SHALL be bilaterally agreed upon. When the "Certificate Exchange Messaging for EDIINT" specification is widely implemented by software vendors, its use will be strongly recommended. This IETF specification will enable automated certificate exchange once the initial trust relationship is established, and will significantly reduce the operational burden of manually exchanging certificates prior to their expiration.

Recommendation 14 – HTTP and HTTP/S Port Numbers for AS2

Receiving AS2 messages on a single port (for each protocol) significantly minimises operational complexities such as firewall set-up for both the sending and receiving partner. Ideally, all AS2 partners would receive messages using the same port number. However some AS2 partners have previously standardised to use a different port number than others and changing to a new port number would add costs without commensurate benefits.

Therefore AS2 partners MAY standardise on the use of port 4080 to receive HTTP messages and the use of port 5443 to receive HTTP/S (SSL) messages.

Recommendation 15 – Duplicate AS2 Messages

AS2 software implementations SHOULD use the 'AS2 Message-ID' value to detect duplicate messages and avoid sending the payload from the duplicate message to internal business applications. The Receiver of Message SHALL return an appropriate MDN even when a message is detected as a duplicate. Note: The Internet Engineering Task Force (IETF) is developing an "Operational Reliability for EDIINT AS2" specification which defines procedures to avoid duplicates and ensure reliability.

Recommendation 15 – Technical Support

There SHOULD be a technical support contact for each Sender of Message and Receiver of Message. The contact information SHOULD include name, email address and phone number. For 24x7x365 operation, a pager or help desk information SHOULD be also provided.

13.4 Bindings for query callback interface

This section specifies bindings for the Query Callback Interface. Each binding includes a specification for a URI that may be used as the `dest` parameter to the `subscribe` method of § *8.2.5*. Each subsection below specifies the conformance requirement (MAY, SHOULD, SHALL) for each binding.

Implementations MAY support additional bindings of the Query Callback Interface. Any additional binding SHALL NOT use a URI scheme already used by one of the bindings specified herein.

All destination URIs, whether standardised as a part of this specification or not, SHALL conform to the general syntax for URIs as defined in [RFC2396]. Each binding of the Query Callback Interface may impose additional constraints upon syntax of URIs for use with that binding.

13.4.1 General Considerations for all XML-based bindings

The following applies to all XML-based bindings of the Query Callback Interface, including the bindings specified in § *13.4.2*, *13.4.3*, and *13.4.4.*

The payload delivered to the recipient SHALL be an XML document conforming to the schema specified in § 13.1. Specifically, the payload SHALL be an `EPCISQueryDocument` instance whose `EPCISBody` element contains one of the three elements shown in the table below, according to the method of the Query Callback Interface being invoked:

Query Callback Interface Method	Payload Body Contents
`callbackResults`	`QueryResults`
`callbackQueryTooLargeException`	`QueryTooLargeException`
`callbackImplementationException`	`ImplementationException`

In all cases, the `queryName` and `subscriptionID` fields of the payload body element SHALL contain the `queryName` and `subscriptionID` values, respectively, that were supplied in the call to `subscribe` that created the standing query.

13.4.2 HTTP binding of the query callback interface

The HTTP binding provides for delivery of standing query results in XML via the HTTP protocol using the POST operation. Implementations MAY provide support for this binding.

The syntax for HTTP destination URIs as used by EPCIS SHALL be as defined in [RFC2616], § 3.2.2. Informally, an HTTP URI has one of the two following forms:

http://host:port/remainder-of-URL
http://host/remainder-of-URL

where

- *`host`* is the DNS name or IP address of the host where the receiver is listening for incoming HTTP connections.
- *`port`* is the TCP port on which the receiver is listening for incoming HTTP connections. The port and the preceding colon character may be omitted, in which case the port SHALL default to 80.
- *`remainder-of-URL`* is the URL to which an HTTP POST operation will be directed.

The EPCIS implementation SHALL deliver query results by sending an HTTP POST request to receiver designated in the URI, where *`remainder-of-URL`* is included in the HTTP `request-line` (as defined in [RFC2616]), and where the payload is an XML document as specified in § 13.4.1.

The interpretation by the EPCIS implementation of the response code returned by the receiver is outside the scope of this specification; however, all implementations SHALL interpret a response code 2xx (that is, any response code between 200 and 299, inclusive) as a normal response, not indicative of any error.

13.4.3 HTTPS binding of the query callback interface

The HTTPS binding provides for delivery of standing query results in XML via the HTTP protocol using the POST operation, secured via TLS. Implementations MAY provide support for this binding.

The syntax for HTTPS destination URIs as used by EPCIS SHALL be as defined in [RFC2818], § 2.4, which in turn is identical to the syntax defined in [RFC2616], § 3.2.2, with the substitution of `https` for `http`. Informally, an HTTPS URI has one of the two following forms:

https://host:port/remainder-of-URL
https://host/remainder-of-URL

where

- *host* is the DNS name or IP address of the host where the receiver is listening for incoming HTTP connections.
- *port* is the TCP port on which the receiver is listening for incoming HTTP connections. The port and the preceding colon character may be omitted, in which case the port SHALL default to 443.
- *remainder-of-URL* is the URL to which an HTTP POST operation will be directed.

The EPCIS implementation SHALL deliver query results by sending an HTTP POST request to receiver designated in the URI, where *remainder-of-URL* is included in the HTTP `request-line` (as defined in [RFC2616]), and where the payload is an XML document as specified in § 13.4.1.

For the HTTPS binding, HTTP SHALL be used over TLS as defined in [RFC2818]. TLS for this purpose SHALL be implemented as defined in [RFC2246] except that the mandatory cipher suite is `TLS_RSA_WITH_AES_128_CBC_SHA`, as defined in [RFC3268] with CompressionMethod.null. Implementations MAY support additional cipher suites and compression algorithms as desired

The interpretation by the EPCIS implementation of the response code returned by the receiver is outside the scope of this specification; however, all implementations SHALL interpret a response code 2xx (that is, any response code between 200 and 299, inclusive) as a normal response, not indicative of any error.

13.4.4 AS2 Binding of the query callback interface

The AS2 binding provides for delivery of standing query results in XML via AS2 [RFC4130]. Implementations MAY provide support for this binding.

The syntax for AS2 destination URIs as used by EPCIS SHALL be as follows:

`as2:`*remainder-of-URI*

where

- *remainder-of-URI* identifies a specific AS2 communication profile to be used by the EPCIS Service to deliver information to the subscriber. The syntax of *remainder-of-URI* is specific to the particular EPCIS Service to which the subscription is made, subject to the constraint that the complete URI SHALL conform to URI syntax as defined by [RFC2396].

Typically, the value of *remainder-of-URI* is a string naming a particular AS2 communication profile, where the profile implies such things as the HTTP URL to which AS2 messages are to be delivered, the security certificates to use, etc. A client of the EPCIS Query Interface wishing to use AS2 for delivery of standing query results must pre-arrange with the provider of the EPCIS Service the specific value of *remainder-of-URI* to use.

Non-Normative: Explanation: Use of AS2 typically requires pre-arrangement between communicating parties, for purposes of certificate exchange and other out-of-band negotiation as part of a bilateral trading partner agreement (see [RFC4130] § *5.1*). The `remainder-of-URI` part of the AS2 URI essentially is a name referring to the outcome of a particular pre-arrangement of this kind.

The EPCIS implementation SHALL deliver query results by sending an AS2 message in accordance with [RFC4130]. The AS2 message payload SHALL be an XML document as specified in § *13.4.1.*

Both the EPCIS Service and the recipient of standing query results SHALL comply with the Requirements and SHOULD comply with the Recommendations listed in the GS1 document "EDIINT AS1 and AS2 Transport Communications Guidelines" [EDICG] For reference, the relevant portions of this document are reproduced in § *13.3.*

14 Conformance

The EPCIS standard defines both standard event data and standard interfaces between system components that communicate event data. Both the data formats and the interfaces may be implemented by a variety of software and data components in any given system.

This section defines what it means to conform to the EPCIS standard. As there are many types of system components that have the potential to conform to various parts of the EPCIS standard, they are enumerated below. In the text that follows, any reference to a section of the EPCIS standard should be understood to refer to that section in its entirety, including subsections thereof.

14.1 Conformance of EPCIS XML data

An electronic document is in conformance to the EPCIS standard when all of the following are true:

- The document is a well-formed XML document conforming to [XML1.0].
- The document conforms to the XML schema for `EPCISDocument` specified in § 9.5 or the XML schema for `EPCISQueryDocument` specified in § *13.1*, as well as all additional constraints specified in the respective section.
- All EPCIS event data within the document (if any) conforms to the definitions of EPCIS event data specified in § 7 and its subsections.
- All master data within the document (if any) conforms to the constraints upon master data specified in § 6.1.1 and 9.4.
- All uses of the extension mechanism (if any) conform to the constraints specified in § *9.1.*
- If a Standard Business Document Header is present, it conforms to the constraints specified in § *9.2*.

Many applications of EPCIS will require, in addition to conformance to the EPCIS standard, that data conform to the EPCIS Comprehensive Business Vocabulary [CBV2.0] standard. The CBV standard defines two conformance levels termed "CBV Compliant" and "CBV Compatible". See the CBV standard for details.

14.2 Conformance of EPCIS capture interface clients

A system is in conformance to the EPCIS standard as a capture interface client when all of the following are true:

- The system conforms to all statements appearing in either § *11.1* or § *11.2* that are indicated as pertaining to a "capture client."

Such a system is said to conform to a particular binding of the capture interface (or more than one binding) depending on which subsection of Section 11 it conforms to.

14.3 Conformance of EPCIS capture interface servers

A system is in conformance to the EPCIS standard as a capture interface server when all of the following are true:

- The system conforms to the statements appearing in § *8.1*.
- The system conforms to all statements appearing in either § *11.1* or § *11.2* that are indicated as pertaining to a "capture server."
- The system processes the `recordTime` field in EPCIS events as specified in the table in § *7.4.1*.

Such a system is said to conform to a particular binding of the capture interface (or more than one binding) depending on which subsection of § 11 it conforms to.

14.4 Conformance of EPCIS query interface clients

A system is in conformance to the EPCIS standard as a query interface client when either or both of the following are true:

- The system conforms to the definition of a "sender" as specified in [WSI] and sends messages in conformance to the WSDL specification in § 13.2.
- The system conforms to all statements appearing in § *13.3* that are indicated as pertaining to a "query client."

Such a system is said to conform to a particular binding of the query interface (or more than one binding) depending on which subsection of § 12 it conforms to.

14.5 Conformance of EPCIS query interface servers

A system is in conformance to the EPCIS standard as a query interface server when all of the following are true:

- The system conforms to the statements appearing in § 8.2.
- The system includes the `recordTime` field in all EPCIS events returned as query results, as specified in the table in § *7.4.1*.

- One or both of the following are true:
  - The system conforms to the definition of a "receiver" as specified in [WSI], receives messages in conformance to the WSDL specification in § *11.2*, and also conforms to the additional constraints specified in § 13.2.
  - The system conforms to all statements appearing in § *13.3* that are indicated as pertaining to a "query server."

Such a system is said to conform to a particular binding of the query interface (or more than one binding) depending on which subsection of § 12 it conforms to.

14.6 Conformance of EPCIS query callback interface implementations

A system is in conformance to the EPCIS standard as a query callback interface implementation when it conforms to the statements appearing in one or more subsections of § 13.4. Such a system is said to conform to a particular binding of the query callback interface (or more than one binding) depending on which subsection it conforms to.

15 References

Normative references:

[ALE] EPCglobal, "The Application Level Events (ALE) Specification, Version 1.1.1," EPCglobal Standard Specification, March 2009, http://www.gs1.org/ale.

[CBV2.0] GS1, "GS1 Comprehensive Business Vocabulary (CBV) Version 2.02 Standard," GS1 standard, 2021, http://www.gs1.org/epcis.

[CBV1.2.2] GS1, "GS1 Core Business Vocabulary (CBV) Version 1.2.2 Standard," GS1 standard, October 2017, http://www.gs1.org/epcis.

[CBV CN 17-339] GS1, Core Business Vocabulary Change Notification (CBVCN), "*Addition of tax ID to the CBV,*" March 2018, http://www.gs1.org/epcis.

[CBV CN 18-108] GS1, Core Business Vocabulary Change Notification (CBVCN), "*Addition of attributes for fish traceability to the CBV,*" July 2018, http://www.gs1.org/epcis.

[CEFACT20] United Nations Economic Commission for Europe, "Recommendation 20: Codes for Units of Measure Used in International Trade," September 2010, http://www.unece.org/fileadmin/DAM/cefact/recommendations/rec20/rec20_Rev7e_2010.zip.

[CRON] https://pubs.opengroup.org/onlinepubs/007904975/utilities/crontab.html

[CURIE] Compact URI Expressions, https://www.w3.org/TR/curie/

[EDICG] GS1, "EDIINT AS1 and AS2 Transport Communications Guidelines," GS1 Technical Document, February 2006, http://www.gs1.org/gs1-xml/guideline/ediint-as1-and-as2-transport-communication-guidelines.

[DCTERMS] Dublin Core Metadata Inititative Terms, http://purl.org/dc/terms/

[EPCIS1.2] https://www.gs1.org/sites/default/files/docs/epc/EPCIS-Standard-1.2-r-2016-09-29.pdf

[GS1UC] "Unit Converter UNECE Rec 20", JavaScript library for converting quantitative values expressed using UN ECE Recommendation 20 unit codes, https://github.com/gs1/UnitConverterUNECERec20

[HTML] https://html.spec.whatwg.org/

[HTTP] https://tools.ietf.org/html/rfc2616

[HTTPHeaders] https://www.iana.org/assignments/message-headers/message-headers.xhtml

[HTTPSemanticsContent] https://datatracker.ietf.org/doc/html/rfc2616#section-12

IETF RFC 8259 [https://datatracker.ietf.org/html/rfc8259]

ISO/IEC 21778:2017

[ISODir2] ISO, "Rules for the structure and drafting of International Standards (ISO/IEC Directives, Part 2, 2018, 8th edition)," https://www.iso.org/sites/directives/current/part2/index.xhtml ; specifically § 7, "*Verbal forms for expressions of provisions*", https://www.iec.ch/standardsdev/resources/draftingpublications/directives/principles/verbal_forms.htm .

[JSON] JavaScript Object Notation, https://www.json.org

[JSON-LD] JSON for Linked Data, https://json-ld.org, https://www.w3.org/TR/json-ld/

JSON Schema: https://json-schema.org/specification.html

[MQTT] https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html#_Toc3901236

[OpenAPI] https://swagger.io/specification/

[OWL] Web Ontology Language (OWL), https://www.w3.org/TR/owl2-syntax/

[RDFS] RDF Schema (RDFS), https://www.w3.org/TR/rdf-schema/

[REST] https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

[RFC1738] T. Berners-Lee, L. Masinter, M. McCahill, "Uniform Resource Locators (URL)," RFC 1738, December 1994, https://datatracker.ietf.org/html/rfc1738.

[RFC8141] R. Moats, "URN Syntax," Internet Engineering Task Force Request for Comments RFC-8141, May 1997, https://datatracker.ietf.org/html/rfc8141.

[RFC2246] T. Dierks, C. Allen, "The TLS Protocol, Version 1.0," RFC2246, January 1999, https://datatracker.ietf.org/html/rfc2246.

[RFC2396] T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax," RFC2396, August 1998, https://datatracker.ietf.org/html/rfc2396.

[RFC2616] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1," RFC2616, June 1999, https://datatracker.ietf.org/html/rfc2616.

[RFC2818] E. Escorla, "HTTP Over TLS," RFC2818, May 2000, https://datatracker.ietf.org/html/rfc2818.

[RFC3268] P. Chown, "Advanced Encryption Standard (AES) Cipersuites for Transport Layer Security (TLS)," RFC3268, June 2002, https://datatracker.ietf.org/html/rfc3268.

[RFC3986] https://datatracker.ietf.org/html/rfc3986

[RFC4122] https://datatracker.ietf.org/html/rfc4122

[RFC4130] D. Moberg and R. Drummond, "MIME-Based Secure Peer-to-Peer Business Data Interchange Using HTTP, Applicability Statement 2 (AS2)," RFC4130, July 2005, *https://datatracker.ietf.org/html/rfc4130*.

[RFC6455] https://datatracker.ietf.org/html/rfc6455

[RFC7231] https://datatracker.ietf.org/html/rfc7231

[RFC7235] *https://datatracker.ietf.org/html/rfc7235*

[RFC7303] *https://datatracker.ietf.org/html/rfc7303*

[RFC7807] *https://datatracker.ietf.org/html/rfc7807*

[RFC8288] *https://datatracker.ietf.org/html/rfc8288*

[SBDH] United Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT), "Standard Business Document Header Technical Specification, Version 1.3," June 2004, http://www.gs1.org/services/gsmp/kc/ecom/xml/xml_sbdh.html

[SHACL] Shapes Constraint Language (SHACL), W3C Recommendation of 20 July 2017, https://www.w3.org/TR/shacl/

[SKOS] Simple Knowledge Organization System, *https://www.w3.org/TR/skos-primer/*

[Snowflake] https://developer.twitter.com/en/docs/basics/twitter-ids.html

[SOAP] https://www.w3.org/TR/soap/

[SPARQL] SPARQL Protocol and RDF Query Language, *https://www.w3.org/TR/sparql11-query/*

[TDS] GS1, "GS1 EPCglobal Tag Data Standard" (TDS), GS1 standard, *http://www.gs1.org/epc/tag-data-standard*

[Turtle] Terse RDF Triple Notation, *https://www.w3.org/TR/turtle/*

[URL] https://url.spec.whatwg.org/

[WebSocket] *https://datatracker.ietf.org/html/rfc6455*

[WSDL1.1] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana, "Web Services Description Language (WSDL) 1.1," W3C Note, March 2001, *http://www.w3.org/TR/2001/NOTE-wsdl-20010315*.

[WSI] K. Ballinger, D. Ehnebuske, M. Gudgin, M. Nottingham, P. Yendluri, "Basic Profile Version 1.0," WS-i Final Material, April 2004, *http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html*.

[XML] https://www.w3.org/XML/

[XML1.0] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau, "Extensible Markup Language (XML) 1.0 (Third Edition)," W3C Recommendation, February 2004, *http://www.w3.org/TR/2004/REC-xml-20040204/*.

[XMLDR] "XML Design Rules for EAN.UCC, Version 2.0," February 2004.

[XMLVersioning] D. Orchard, "Versioning XML Vocabularies," December 2003, http://www.xml.com/pub/a/2003/12/03/versioning.html.

[XSD1] H. Thompson, D. Beech, M. Maloney, N. Mendelsohn, "XML Schema Part 1: Structures," W3C Recommendation, May 2001, http://www.w3.org/TR/xmlschema-1/.

[XSD2] P. Biron, A. Malhotra, "XML Schema Part 2: Datatypes," W3C Recommendation, May 2001, http://www.w3.org/TR/xmlschema-2/.

Non-normative references:

[GS1Arch] "The GS1 System Architecture," GS1 technical document, Release 9, February 2020,

http://www.gs1.org/docs/gsmp/architecture/GS1_System_Architecture.pdf

[EPCISGuideline] GS1, "EPCIS and CBV Implementation Guideline," GS1 Guideline, February 2017, http://www.gs1.org/docs/epc/EPCIS_Guideline.pdf.

16 Contributors to earlier versions

16.1 Contributors to EPCIS 1.2

Below is a list of more active participants and contributors in the development of EPCIS 1.2.

Name	Company
Andrew Kennedy, Co-chair	FoodLogiQ
Ralph Troeger, Co-chair	GS1 Germany
Gena Morgan, Facilitator	GS1 Global Office
Ken Traub, Editor	Ken Traub Consulting LLC
Philip Allgaier	bpcompass GmbH
Paul Arguin	r-pac international
Karla Biggs-Gregory	Oracle
Zsolt Bocsi	GS1 Hungary
Jonas Buskenfried	GS1 Sweden
Jaewook Byun	Auto-ID Labs, KAIST
Karolin Catela	GS1 Sweden
Mario Chavez	GS1 Guatemala
Luiz Costa	GS1 Brasil
Deniss Dobrovolskis	GS1 Sweden

Name	Company
Michael Dols	MET Laboratories
Hussam El-Leithy	GS1 US
Jürgen Engelhardt	Robert Bosch GmbH
Heinz Graf	GS1 Switzerland
Danny Haak	Nedap
Tany Hui	GS1 Hong Kong, China
Jianhua Jia	GS1 China
Peter Jonsson	GS1 Sweden
Art Kaufmann	Frequentz LLC
Janice Kite	GS1 Global Office
Jens Kungl	METRO Group
Roar Lorvik	GS1 Norway
Paul Lothian	Tyson
Fargeas Ludovic	Courbon
Noriyuki Mama	GS1 Japan
Kevan McKenzie	McKesson
Reiko Moritani	GS1 Japan
Alice Mukaru	GS1 Sweden
Mauricio Munoz	Axway
Falk Nieder	EECC
Juan Ochoa	GS1 Columbia
Ted Osinski	MET Laboratories
Ben Östman	GS1 Finland
James Perng	GS1 Chinese Taipei
Craig Alan Repec	GS1 Global Office
Chris Roberts	GlaxoSmithKline
Thomas Rumbach	SAP AG
Chuck Sailer	Frequentz

Name	Company
Michael Sarachman	GS1 Global Office
Hans Peter Scheidt	GS1 Germany
Michael Smith	Merck & Co., Inc.
Michele Southall	GS1 US
Peter Spellman	TraceLink
Peter Sturtevant	GS1 US
Hristo Todorov	Axway
Geir Vevle	HRAFN AS
Elizabeth Waldorf	TraceLink
Ruoyun Yan	GS1 China
Tony Zhang	FSE, Inc.
Mike Zupec	Abbvie

16.2 Contributors to EPCIS 1.1

Below is a list of active participants and contributors in the development of EPCIS 1.1.

Name	Company
Andrew Kennedy	FoodLogiQ (Working group co-chair)
Michele Southall	GS1 US (Working group co-chair)
Gena Morgan	GS1 (Working group facilitator)
Ken Traub	Ken Traub Consulting LLC (Editor)
Craig Alan Repec	GS1 Global Office
Jean-Pierre Allard	Optel Vision
Romain Arnaud	Courbon
Shirley Arsenault	GS1 Global Office
Koji Asano	GS1 Japan
Karla Biggs-Gregory	Oracle

Name	Company
Havard Bjastad	TraceTracker AS
Stephan Bourguignon	Daimler AG
Bob Bunsey	SPEDE Technologies
Birgit Burmeister	Daimler AG
Jonas Buskenfried	GS1 Sweden
Robert Celeste	GS1 US
Chris Chandler	GS1 US
Lucy Deus	Tracetracker
Hussam El-Leithy	GS1 US
Heinz Graf	GS1 Switzerland
Anders Grangard	GS1 Global
Emmanuel Hadzipetros	TraceLink
Mark Harrison	Auto-ID Labs
Dave Harty	Systech International
Douglas Hill	GS1 Denmark
Robert Hotaling	Supply Insight
John Howells	HDMA
Tany Hui	GS1 Hong Kong
Yoshihiko Iwasaki	GS1 Japan
Art Kaufmann	Frequentz LLC, IBM
John Keogh	GS1 Global Office
Janice Kite	GS1 Global Office
Steinar Kjærnsrød	TraceTracker AS
Jay Kolli	Abbott
Jens Kungl	METRO Group
Sean Lockhead	GS1 Global Office
Paul Lothian	Tyson
Dale Moberg	Axway

Name	Company
Reiko Moritani	GS1 Japan
Mark Morris	Abbott Laboratories Inc.
Marc-Antoine Mouilleron	France Telecom Orange
Alice Mukaru	GS1 Sweden
Falk Nieder	IBM Germany
Andrew Osbourne	GS1 UK
Ted Osinski	MET Laboratories
Nicolas Pauvre	GS1 France
Cynthia Poetker	Abbott Laboratories
Venkataramanan Rajaraman	Abbott Laboratories
Craig Alan Repec	GS1 Global Office
Chris Roberts	GlaxoSmithKline
Ian Robertson	Supply Chain RFID Consulting LLC
Dirk Rodgers	Dirk Rodgers Consulting, LLC
Thomas Rumbach	SAP AG
John Ryu	GS1 Global Office
Aravindan Sankaramurthy	Oracle
Michael Sarachman	GS1 Global Office
Udo Scheffer	METRO Group
Frank Schmid	IBM (US)
Michael Smith	Merck & Co., Inc.
Monika Solanki	Aston University
Peter Spellman	TraceLink
Steve Tadevich	McKesson
Petter Thune-Larsen	GS1 Norway
Peter Tomicki	Zimmer, Inc.
Ralph Troeger	GS1 Germany
Jens Vialkowitsch	Robert Bosch GmbH

Name	Company
Geir Vevle	HRAFN
Matthew Warren	Zimmer, Inc.
David Weatherby	GS1 UK
Joachim Wilkens	C & A SCS

3343

3344 ## 16.3 Contributors to EPCIS 1.0

3345 Below is a list of more active participants and contributors in the development of EPCIS 1.0.

Name	Company
Craig Asher	IBM (Co-Chair)
Greg Gibert	Verisign (Co-Chair)
Richard Swan	T3Ci (Co-Chair)
Ken Traub	BEA Systems; ConnecTerra (Specification Editor)
Gena Morgan	EPCglobal, Inc. (WorkGroup Facilitator)
Chi-Hyeong Ahn	Ceyon Technology Co., Ltd
Umair Akeel	IBM
John Anderla	Kimberly-Clark Corp
Richard Bach	Globe Ranger
Scott Barvick	Reva Systems
Sylvanus Bent	Bent Systems, Inc.
Hersh Bhargava	Rafcor
Chet Birger	ConnecTerra
Bud Biswas	Polaris Networks
Prabhudda Biswas	Oracle Corporation
Havard Bjastad	Tracetracker
Joe Bohning	Nestle Purina

Name	Company
Al Bottner	UNITED PARCEL SERVICE (UPS)
Joe Bradley	Sun Microsystems
Leo Burstein	Gillette; Procter & Gamble
Anit Chakraborty	Oracle Corporation
Chia Chang	Sun Microsystems
Ying-Hung Chang	Acer Cybercenter Service Inc.
Martin Chen	SAP
Nagesh Chigurupati	VeriSign
Christian Clauss	IBM
John Cooper	Kimberly-Clark Corp
Valir-Alin Crisan	IBM
Mustafa Dohadwala	Shipcom Wireless, Inc.
John Duker	Procter & Gamble
Igor Elbert	Sensitech
Ronny Fehling	Oracle Corporation
Akira Fujinami	Internet Initiative Japan, Inc.
Tony Gallo	Real Time Systems
Manish Gambhir	
Cesar Gemayel	Sensitech
Eric Gieseke	BEA Systems
Greg Gilbert	Manhattan Associates
Graham Gillen	Verisign
John Gravitis	Allumis
Yuichiro Hanawa	Mitsui
Mark Harrison	Auto-ID Labs - Cambridge
Jeremy Helm	ACSIS
Barba Hickman	Intermec
Manju James	BEA Systems

Name	Company
Paul Jatkowski	
Jennifer Kahn	IBM
Howard Kapustein	Manhattan Associates
Sean Lockhead	GS1 US
Paul Lovvik	Sun Microsystems
Midori Lowe	Nippon Telegraph & Telephone Corp (NTT)
Dave Marzouck	SAP
Andrew McGrath	Manhattan Associates
Michael Mealling	Verisign; Refactored Networks
Stephen Miles	Auto-ID Labs - MIT
Tim Milne	Target
Dale Moberg	AXWAY/formerly Cyclone
Stephen Morris	Printronix
Ron Moser	Wal-Mart
Don Mowery	Nestle
Doug Naal	Altria Group, Inc./Kraft Foods
David Nesbitt	Vue Technology
Shigeki Ohtsu	Internet Initiative Japan, Inc.
Ted Osinski	MET Labs
Jong Park	Tibco
Ju-Hyun Park	Samsung SDS
Sung Gong Park	Metarights
Eliot Polk	Reva Systems
Mike Profit	Verisign
Sridhar Ramachandran	OAT Systems
Ajay Ramachandron	
Karen Randall	Johnson & Johnson
Steve Rehling	Procter & Gamble

Name	Company
Nagendra Revanur	T3Ci Incorporated
Thomas Rumbach	SAP
Uday Sadhukhan	Polaris Networks
Hares Sangani	Hubspan, Inc.
Puneet Sawhney	CHEP
Rick Schendel	Target
Chris Shabsin	BEA Systems
Bhavesh Shah	Abbott Laboratories
Harshal Shah	Oracle Corporation
Dong Cheul Shin	Metarights
Sung-hak Song	Samsung SDS
Ashley Stephenson	Reva Systems
Nikola Stojanovic	GS1 US
Jim Sykes	Savi Technology
Hiroki Tagato	NEC Corporation
Diane Taillard	GS1 France
Neil Tan	UPS
Zach Thom	Unilever
Frank Thompson	Afilias Canada Corp
Frank Tittel	Gedas Deutschland GmbH
Bryan Tracey	Globe Ranger
Hsi-Lin Tsai	Acer Cybercenter Service Inc.
Richard Ulrich	Walmart
David Unge	
Steve Vazzano	1Sync
Vasanth Velusamy	Supply Insight, Inc.
Dan Wallace	
Jie Wang	True Demand Software (fka-Truth Software)

Name	Company
John Williams	Auto-ID Labs - MIT
Michael Williams	Hewlett-Packard Co. (HP)
Steve Winkler	SAP
Katsuyuki Yamashita	Nippon Telegraph & Telephone Corp (NTT)
Patrick Yee	Hubspan, Inc.
Angela Zilmer	Kimberly-Clark Corp

3346