INTERNATIONAL STANDARD

ISO/IEC 19987

First edition
2015-09-15

Information technology — EPC Information services — Specification

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

Reference number
ISO/IEC 19987:2015(E)

ISO copyright office
Ch. de Blandonnet 8 • CP 401
CH-1214 Vernier, Geneva, Switzerland
Tel. +41 22 749 01 11
Fax +41 22 749 09 47
copyright@iso.org
www.iso.org

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

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

For an explanation on the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the WTO principles in the Technical Barriers to Trade (TBT), see the following URL: Foreword — Supplementary information.

The committee responsible for this document is ISO/IEC JTC 1, *Information technology*.

EPC Information Services (EPCIS) Version 1.1 Specification

GS1 Standard

Version 1.1, May 2014

GS1 Global Office
Avenue Louise 326, bte 10
B-1050 Brussels, Belgium

Disclaimer

Abstract

This document is a GS1 Standard that defines Version 1.1 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.

Status of this document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. The latest status of this document series is maintained at GS1. See `www.gs1.org/gsmp` for more information.

This version of the GS1 EPCIS 1.1 Standard is the ratified version and has completed all GSMP steps.

Comments on this document should be sent to `gsmp@gs1.org`.

Differences from EPCIS 1.0.1

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.

Table of Contents

1

Introduction

This document is a GS1 Standard that defines Version 1.1 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 organizations. 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 as of EPCIS 1.1 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 a standard data sharing interface between applications that capture visibility event data and those that need access to it. *It does not specify how the service operations or databases themselves should be implemented.* This includes not defining how the EPCIS services should acquire and/or compute the data they need, except to the extent the data is captured using the standard EPCIS capture operations. The interfaces are needed for interoperability, while the implementations allow for competition among those providing the technology and implementing the standard.

EPCIS is intended to be used in conjunction with the GS1 Core Business Vocabulary (CBV) standard [CBV1.1]. 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 standardized 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.

2 Relationship to the GS1 System Architecture

This section is largely quoted from [EPCAF] and [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.

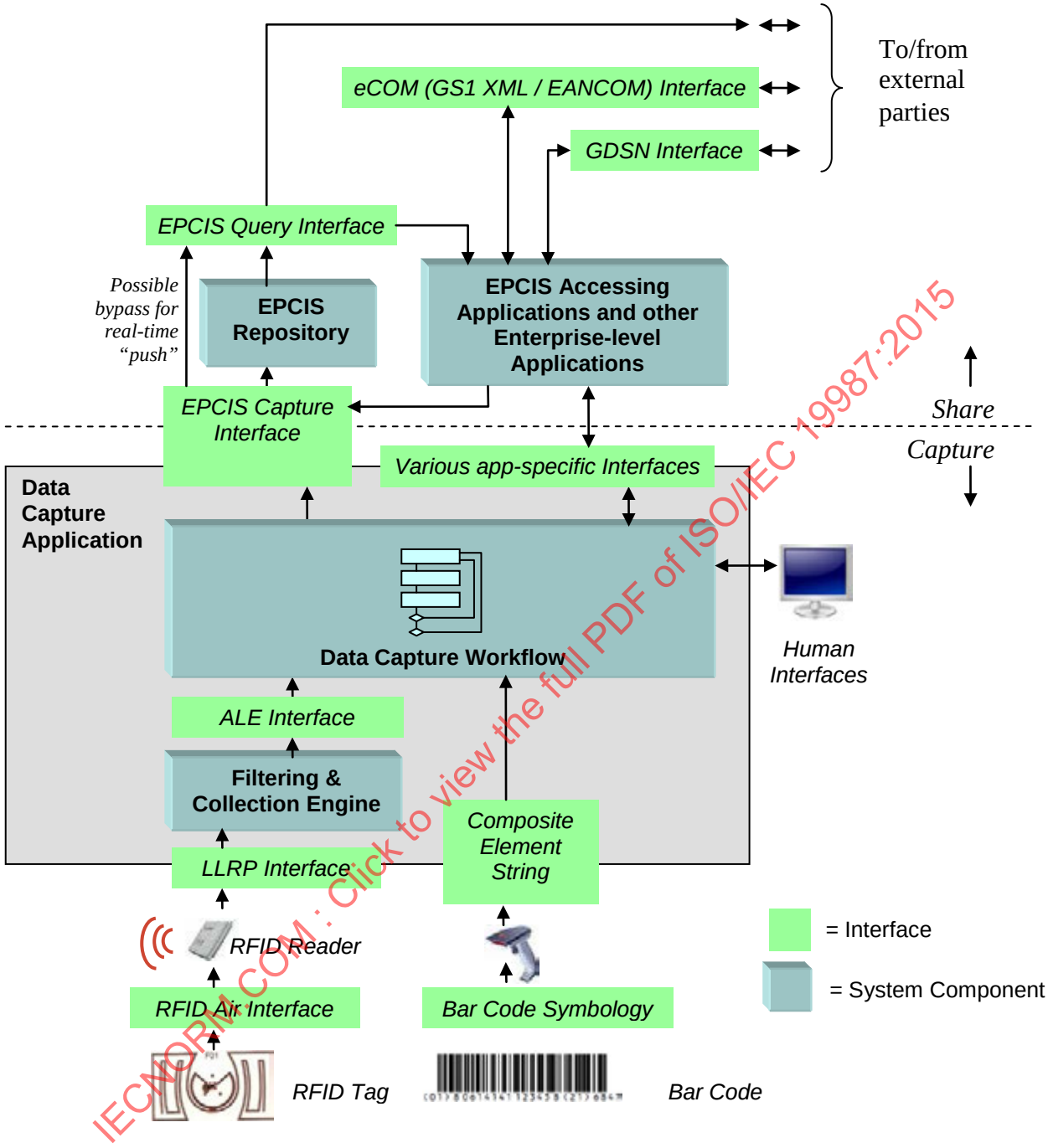
GS1 Standards may be divided into the following groups according to their role in supporting information needs related to real-world entities in supply chain business processes:

- 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 bar code 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 following diagram 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: bar codes 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 bar code 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 bar codes 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 bar code 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).

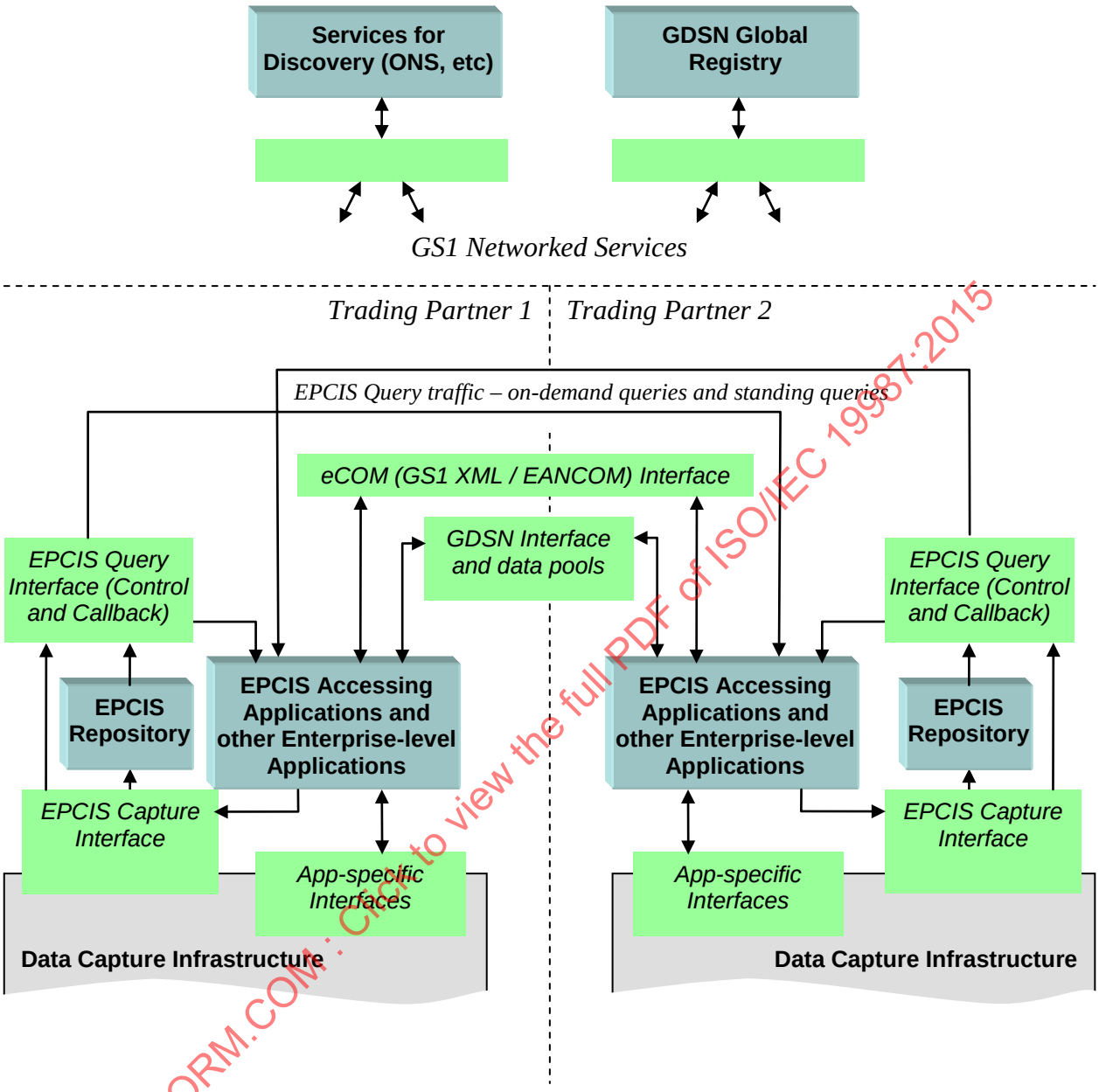
2.3 EPCIS in Relation to Trading Partners

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 eCOM 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 organization's four walls, but across organizations.	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 following figure illustrates the flow of data between trading partners, emphasizing the parts of the EPCIS standard involved in the flow of visibility event data.



267

268 2.4 EPCIS in Relation to other GS1 System Architecture Components

269 The following outlines the responsibilities of each element of the GS1 System Architecture as
270 illustrated in the figures in the preceding sections. Further information may be found in
271 [GS1Arch], from which the above diagram and much of the above text is quoted, and [EPCAF],
272 from which much of the following text is quoted.

273 • *RFID and Bar Code Readers* Make observations of RFID tags while they are in the read
274 zone, and observations of bar codes 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 bar codes, because bar code readers typically only read a single bar code 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 bar codes, comparable data is delivered to the Data Capture Workflow role directly from the bar code 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 bar code 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 bar code 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 bar code 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 bar code 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 actually three EPCIS Interfaces. 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. All three of these EPCIS interfaces are specified normatively in this document.

- *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 EPC-related 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 bar code 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 bar code 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 standardization (as well as of implementation) to scale.

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

4 Terminology and Typographical Conventions

Within this specification, the terms SHALL, SHALL NOT, SHOULD, SHOULD NOT, MAY, NEED NOT, CAN, and CANNOT are to be interpreted as specified in Annex G of the ISO/IEC Directives, Part 2, 2001, 4th 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, with the exception of Sections 1, 2, and 3, 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, and XML identifiers, as well as for the text of XML documents.

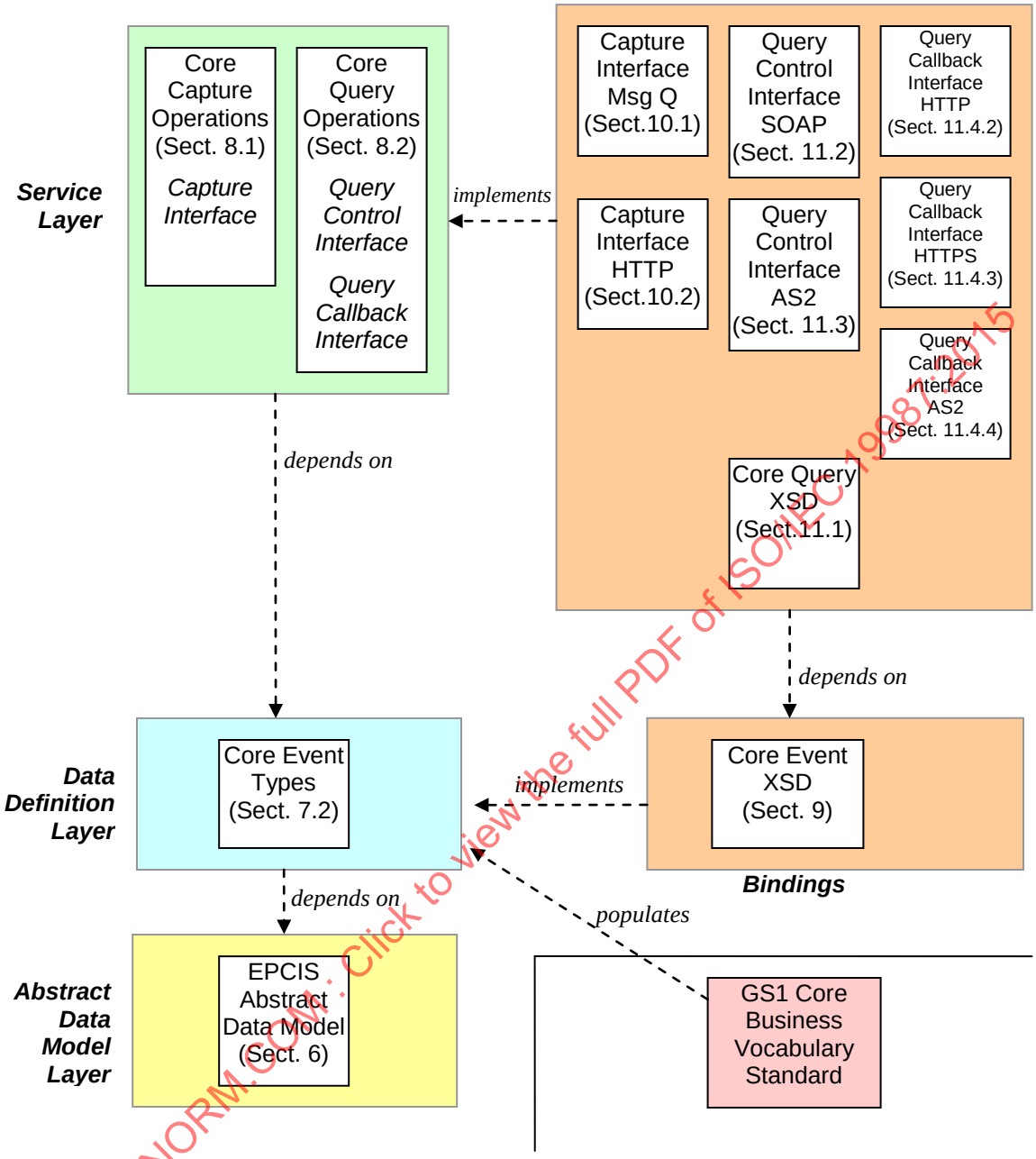
➢ Placeholders for changes that need to be made to this document prior to its reaching the final stage of approved GS1 Standard are prefixed by a rightward-facing arrowhead, as this paragraph is.

5 EPCIS Specification Framework

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

5.1 Layers

The EPCIS specification framework is organized into several layers, as illustrated below:



These layers are 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 a 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 realizations 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 total of nine 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 a binding to an XML schema. The EPCIS Capture Interface in the Core Capture Operations Module is given bindings for Message Queue and HTTP. 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, and a second binding for AS2. The EPCIS Query Callback Interface in the Core Query Operations Module is given bindings to HTTP, HTTPS, and AS2.

- *GS1 Core Business Vocabulary Standard* The GS1 Core Business Vocabulary standard [CBV1.1] 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 denormalize 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

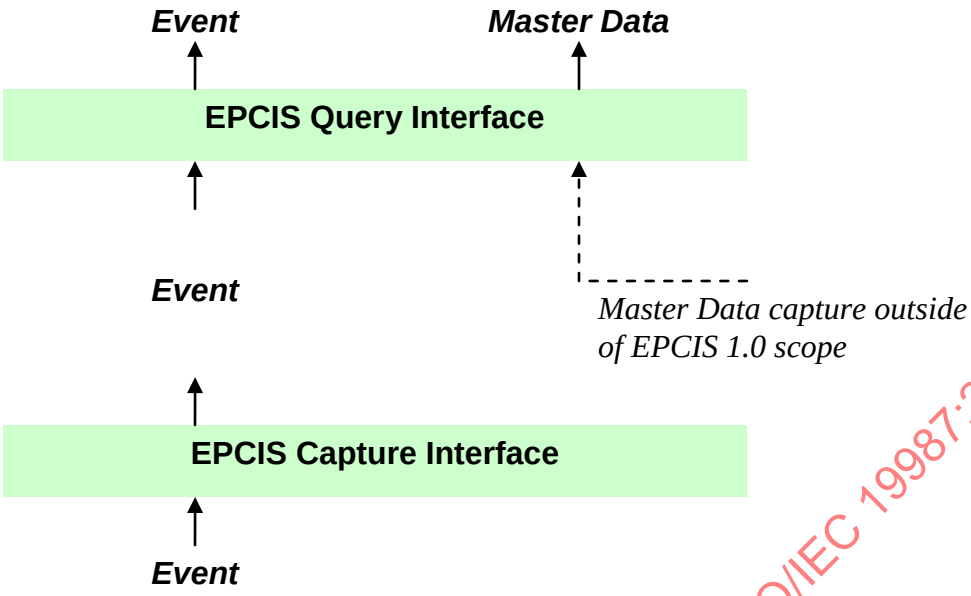
This section gives a normative description of the abstract data model that underlies EPCIS.

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 1.1 specification.

Roadmap (non-normative): It is possible that capture of master data will be addressed in a future version of the EPCIS specification.

These relationships are illustrated below:

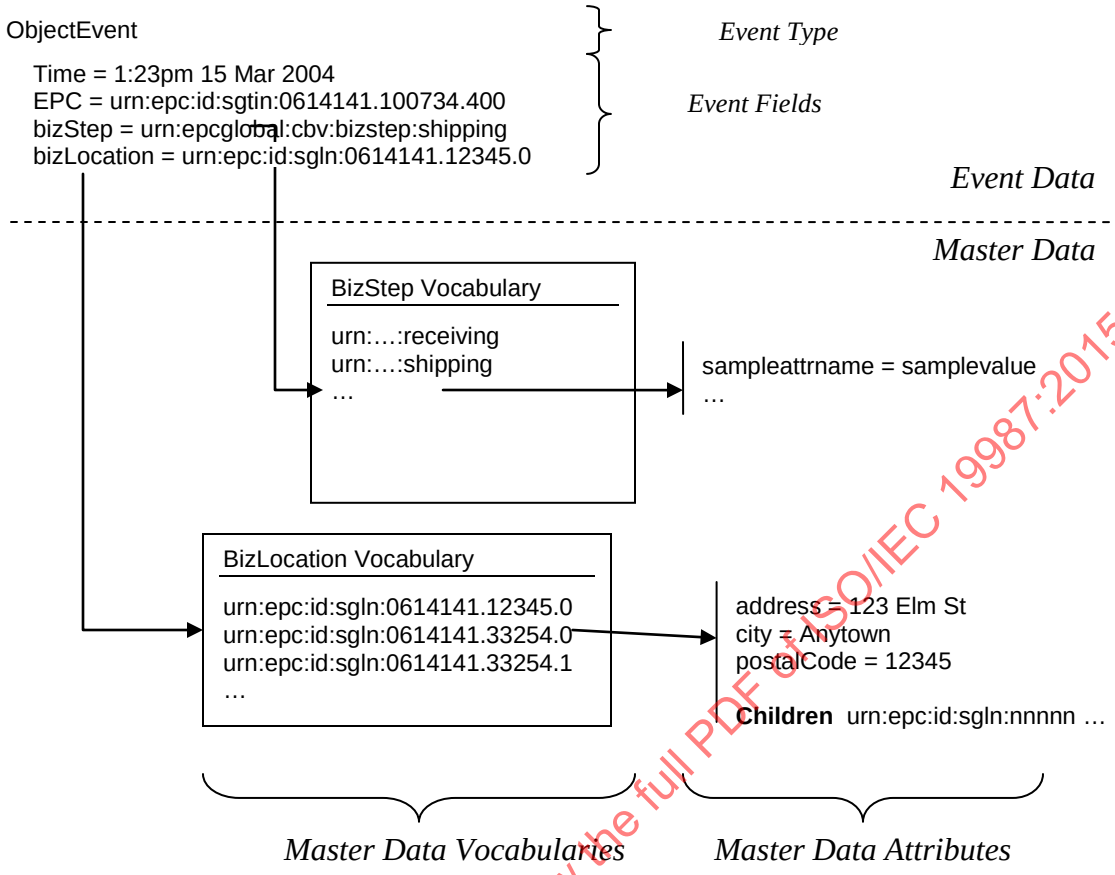


499

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

505 *Explanation (non-normative): While for the purposes of this specification the terms "event data"*
506 *and "master data" mean nothing more than "data that fits the structure provided here," the*
507 *structures defined in the Abstract Data Model Layer are designed to provide an appropriate*
508 *representation for data commonly requiring exchange through EPCIS. Informally, these two*
509 *types of data may be understood as follows. Event data grows in quantity as more business is*
510 *transacted, and refers to things that happen at specific moments in time. An example of event*
511 *data is "At 1:23pm on 15 March 2004, EPC X was observed at Location L." Master data does*
512 *not generally grow merely because more business is transacted (though master data does tend to*
513 *grow as organizations grow in size), is not typically tied to specific moments in time (though*
514 *master data may change slowly over time), and provides interpretation for elements of event*
515 *data. An example of master data is "Location L refers to the distribution centre located at*
516 *123 Elm Street, Anytown, US." All of the data in the set of use cases considered in the creation*
517 *of the EPCIS 1.1 specification can be modelled as a combination of event data and master data*
518 *of this kind.*

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



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

- *Event Data* A set of Events.
- *Event* A structure consisting of an Event Type and one or more named Event Fields.
- *Event Type* A namespace-qualified name (qname) that indicates to which of several possible Event structures (as defined by the Data Definition Layer) a given event conforms.
- *Event Field* A named field within an Event. The name of the field is given by a qname, referring either to a field name specified by the Data Definition Layer or a field name defined as an extension to this specification. The value of the field may be a primitive type (such as an integer or timestamp), a Vocabulary Element, or a list of primitive types or Vocabulary Elements.
- *Master Data* A set of Vocabularies, together with Master Data Attributes associated with elements of those Vocabularies.
- *Vocabulary* A named set of identifiers. The name of a Vocabulary is a qname that may be used as a type name for an event field. The identifiers within a Vocabulary are called Vocabulary Elements. A Vocabulary represents a set of alternative values that may appear as the values of specific Event Fields. Vocabularies in EPCIS are used to model sets such as the set of available location names, the set of available business process step names, and so on.

- *Vocabulary Element* An identifier that names one of the alternatives modelled by a Vocabulary. The value of an Event Field may be a Vocabulary Element. Vocabulary Elements are represented as Uniform Resource Identifiers (URIs). Each Vocabulary Element may have associated Master Data Attributes.

- *Master Data Attributes* An unordered set of name/value pairs associated with an individual Vocabulary Element. The name part of a pair is a qname. The value part of a pair may be a value of arbitrary type. A special attribute is a (possibly empty) list of children, each child being another vocabulary element from the same vocabulary. See Section 6.5.

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

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).

6.2 Vocabulary Kinds

Vocabularies are used extensively with 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 tend to be defined by organizations of multiple end users, such as GS1, industry consortia outside GS1, private trading partner groups, and so on. The master data associated with Standard Vocabulary elements are defined by those same organizations, 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 tend to be introduced through a very deliberate and occasional process, such as the ratification of a new version of a standard or through a vote of an industry group. While an individual end user organization 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 organization'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 organization. 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 individual end user organizations acting independently. The master data associated with User Vocabulary elements are defined by those same organizations, and are usually distributed to trading partners through the EPCIS Query Control Interface or other data exchange / data synchronization mechanisms. New vocabulary elements within a given User Vocabulary are introduced at the sole discretion of an end user, and trading partners must be prepared to respond accordingly. Usually, however, the rules for constructing new User Vocabulary Elements are established by organizations of multiple end users, and in any case must follow the rules defined in Section 6.4 below.

The lines between these two kinds of vocabularies are somewhat subjective. However, the mechanisms defined in the EPCIS specification make absolutely no distinction between the two vocabulary types, and so 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 GS1 Core Business Vocabulary (CBV) standard [CBV1.1] provides standardized 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, Business Transaction Type, and Source/Destination 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, and Transformation ID.

6.3 Extension Mechanisms

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

- *New Event Type* A new Event Type may be added in the Data Definition Layer. Adding a new Event Type requires each of the Data Definition Bindings to be extended, and may also require extension to the Capture and Query Interfaces and their Bindings.
- *New Event Field* A new field may be added to an existing Event Type in the Data Definition Layer. The bindings, capture interface, and query interfaces defined in this specification are designed to permit this type of extension without requiring changes to the specification itself. (The same may not be true of other bindings or query languages defined outside this specification.)
- *New Vocabulary Type* A new Vocabulary Type may be added to the repertoire of available Vocabulary Types. No change to bindings or interfaces are required.
- *New Master Data Attribute* A new attribute name may be defined for an existing Vocabulary. No change to bindings or interfaces are required.
- *New Instance/Lot Master Data (ILMD) Attribute* A new attribute name may be defined for use in Instance/Lot Master Data (ILMD); see Section 7.3.6. 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 summarized below:

How Extension is Disseminated	**Responsible Organization**	**Extension Method**				
		New Event Type	**New Event Field**	**New Vocabulary Type**	**New Master Data or ILMD (Section 7.3.6) Attribute**	**New Vocabulary Element**
New Version of EPCIS standard	GS1 EPCIS Working Group	Yes	Yes	Yes	Occasionally	Rarely
New Version of CBV standard	GS1 Core Business Vocabulary Working Group	No	No	No	Occasionally	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 Disseminated	Responsible Organization	Extension Method				
		New Event Type	New Event Field	New Vocabulary Type	New Master Data or ILMD (Section 7.3.6) Attribute	New Vocabulary Element
GS1 Member Organisation Local Recommendation Document for a specific industry within a specific geography	GS1 Member Organization	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 organizations 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 [TDS1.9, Section 7]. The owning authority for a particular EPC URI is the organization to whom the EPC manager number was assigned.
- Absolute Uniform Resource Locators (URLs) [RFC1738]. The owning authority for a particular URL is the organization that owns the Internet domain name in the authority portion of the URL.

- Uniform Resource Names (URNs) [RFC2141] in the `oid` namespace that begin with a Private Enterprise Number (PEN). The owning authority for an OID-URN is the organization to which the PEN was issued.
- Uniform Resource Names (URNs) [RFC2141] in the `epc` or `epcglobal` namespace, other than URIs used to represent EPCs [TDS1.9]. The owning authority for these URNs is GS1.

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.

Explanation (non-normative): 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 generalize this to allow more than one relationship, perhaps encoding each relationship via a different master data attribute.

Hierarchical relationships are given special treatment in queries (Section 8.2), and may play a role in carrying out authorization policies (Section 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 Section 7.2 to define the Core Event Types Module. These rules can also be applied by organizations wishing to layer a specification on top of this specification.

7.1.1 Content

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

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

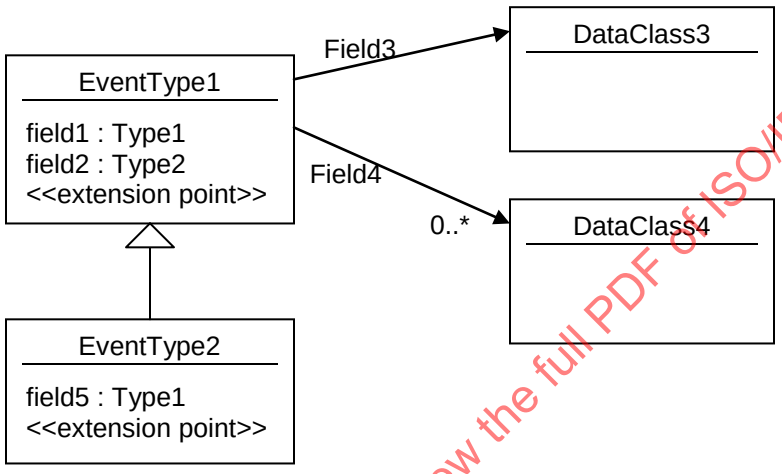
Amplification (non-normative): As explained in Section 6.3, Data Definition Modules defined in this specification and by companion specifications developed by the EPCIS Working Group will tend to include definitions of Value Types, Event Types, Event Fields, and Vocabulary Types, while modules defined by other groups will tend to include definitions of Event Fields that extend

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

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

7.1.2 Notation

In the sections below, Event 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. Here is an example:



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

The Event Type `EventType1` in this example has four fields. `Field1` and `Field2` are of primitive type `Type1` and `Type2` respectively. `EventType1` has another field `Field3` whose type is `DataClass3`. Finally, `EventType1` has another field `Field4` that contains a list of zero or more instances of type `DataClass4` (the "0..*" notation indicates "zero or more").

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

Within the UML descriptions, the notation `<<extension point>>` identifies a place where implementations SHALL provide for extensibility through the addition of new data members. (When one type has an extension point, and another type is defined as a subclass of the first type and also has an extension point, it does not mean the second type has two extension points; rather, it merely emphasizes that the second type is also open to extension.) Extensibility

mechanisms SHALL provide for both proprietary extensions by vendors of EPCIS-compliant products, and for extensions defined by GS1 through future versions of this specification or through new specifications.

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

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

7.1.3 Semantics

Each event (an instance of an Event Type) encodes several assertions which collectively define the semantics of the event. Some of these assertions say what was true at the time the event was captured. Other assertions say what is expected to be true following the event, until invalidated by a subsequent event. These are called, respectively, the *retrospective semantics* and the *prospective semantics* of the event. For example, if widget #23 enters building #5 through door #6 at 11:23pm, then one retrospective assertion is that "widget #23 was observed at door #6 at 11:23pm,", while a prospective assertion is that "widget #23 is in building #5." The key difference is that the retrospective assertion refers to a specific time in the past ("widget #23 *was observed*…"), while the prospective assertion is a statement about the present condition of the object ("widget #23 *is in*…"). The prospective assertion presumes that if widget #23 ever leaves building #5, another EPCIS capture event will be recorded to supersede the prior one.

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

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

7.2 Core Event Types Module – Overview

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

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

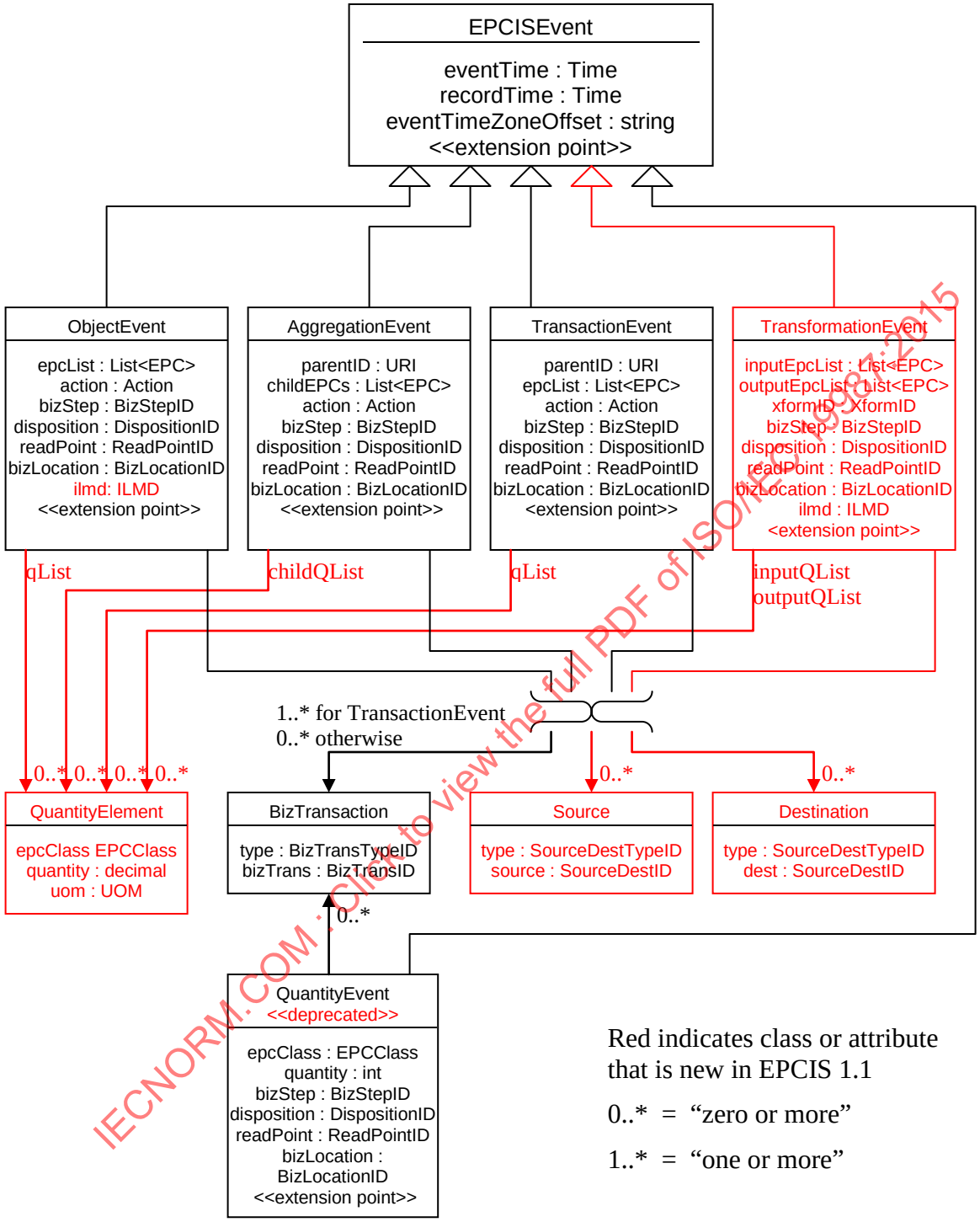
- *Value Types* Primitive types defined in Sections 7.3.1 and 7.3.2.

- *Event Types* Event types as shown in the UML diagram below, and defined in Sections 7.4.1 through 7.4.6.
- *Event Fields* Included as part of the Event Types definitions.
- *Vocabulary Types* Types defined in Sections 7.3.3 through 7.3.5, and summarized 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 (Section 7.3.5), the `Disposition` vocabulary (Section 7.3.5.2), and the `BusinessTransactionType` vocabulary (Section 7.3.5.3.1).

This module defines six event types, one very generic event and five subclasses (one of which is deprecated as of EPCIS 1.1) that can represent events arising from supply chain activity across a wide variety of industries:

- `EPCISEvent` (Section 7.4.1) is a generic base class for all event types in this module as well as others.
- `ObjectEvent` (Section 7.4.2) represents an event that happened to one or more physical or digital objects.
- `AggregationEvent` (Section 7.4.3) represents an event that happened to one or more objects that are physically aggregated together (physically constrained to be in the same place at the same time, as when cases are aggregated to a pallet).
- `QuantityEvent` (Section 7.4.4) represents an event concerned with a specific quantity of objects sharing a common EPC class, but where the individual identities of the entities are not specified. As of EPCIS 1.1, this event is deprecated; an `ObjectEvent` (Section 7.4.2) with one or more `QuantityElements` (Section 7.3.3.3) should be used instead.
- `TransactionEvent` (Section 7.4.5) represents an event in which one or more objects become associated or disassociated with one or more identified business transactions.
- `TransformationEvent` (Section 7.4.6) 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.

A UML diagram showing these Event Types is as follows:



Note: in this diagram, certain names have been abbreviated owing to space constraints; e.g., `BizLocationID` is used in the diagram, whereas the actual type is called `BusinessLocationID`. See the text of the specification for the normative names of fields and their types

839

Each of the core event types (not counting the generic `EPCISEvent`) has fields that represent four key dimensions of any EPCIS event. These four 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. These four dimensions may be conveniently remembered as "what, when, where, and why" (respectively). The "what" dimension varies depending on the event type (e.g., for an `ObjectEvent` the "what" dimension is one or more EPCs; for an `AggregationEvent` the "what" dimension is a parent ID and list of child EPCs). The "where" and "why" dimensions have both a retrospective aspect and a prospective aspect (see Section 7.1.3), represented by different fields.

The following table summarizes the fields of the event types that pertain to the four key dimensions:

	Retrospective **(at the time of the event)**	**Prospective** **(true until contradicted by subsequent event)**
What	`EPC` `EPCClass` + quantity	
When	`Time`	
Where	`ReadPointID`	`BusinessLocationID`
Why (business context)	`BusinessStepID`	`DispositionID`

	`BusinessTransactionList`
	`Source/Destination`
	`ILMD`	

In addition to the fields belonging to the four key dimensions, events may carry additional descriptive information in other fields. It is expected that the majority of additional descriptive information fields will be defined by industry-specific specifications layered on top of this one.

The following table summarizes 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 / Standard**	**URI**
`ReadPointID`	7.3.4	User	`urn:epcglobal:epcis:vtype:ReadPoint`
`BusinessLocationID`	7.3.4	User	`urn:epcglobal:epcis:vtype:BusinessLocation`
`BusinessStepID`	7.3.5	Standard	`urn:epcglobal:epcis:vtype:BusinessStep`
`DispositionID`	7.3.5.2	Standard	`urn:epcglobal:epcis:vtype:Disposition`

Vocabulary Type	Section	User / Standard	URI
`BusinessTransaction`	7.3.5.3.2	User	`urn:epcglobal:epcis:vtype:BusinessTransaction`
`BusinessTrasactionTypeID`	7.3.5.3.1	Standard	`urn:epcglobal:epcis:vtype:BusinessTransactionType`
`EPCClass`	7.3.5.4	User	`urn:epcglobal:epcis:vtype:EPCClass`
`SourceDestTypeID`	7.3.5.4.1	Standard	`urn:epcglobal:epcis:vtype:SourceDestType`
`SourceDestID`	7.3.5.4.2	User	`urn:epcglobal:epcis:vtype:SourceDest`

7.3 Core Event Types Module – Building Blocks

This section specifies the building blocks for the event types defined in Section 7.3.5.4.

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 [TDS1.9]. Unless otherwise noted, EPCs are represented in "pure identity" URI form as defined in [TDS1.9], Section 7.

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

Explanation (non-normative): This reflects a design decision not to consider individual trade item instances as Vocabulary Elements having Master Data, owing to the fact that trade item instances are constantly being created and hence new EPCs representing trade items are constantly being commissioned. In part, this design decision reflects consistent treatment of Master Data as excluding data that grows as more business is transacted (see comment in Section 6.1), and in part reflects the pragmatic reality that data about trade item instances is

likely to be managed more like event data than master data when it comes to aging, database design, etc.

7.3.2 Action Type

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

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

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

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

Note that the three values above are the only three 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

This section defines the data types used in the "What" dimension of the event types specified in Section 7.3.5.4.

7.3.3.1 Instance-level vs. Class-level Identification

The "What" dimension of an EPCIS event specifies what physical or digital objects participated in the event. EPCIS provides for objects to be identified in two ways:

- *Instance-level* An identifier is said to be an instance-level identifier if such identifiers are assigned so that each is unique to a single object. That is, no two objects are allowed to carry the same instance-level identifier.
- *Class-level* An identifier is said to be a class-level identifier if multiple objects may carry the same identifier.

In general, instance-level identifiers allow EPCIS events to convey more information, because it is possible to correlate multiple EPCIS events whose "what" dimension includes the same instance-level identifiers. For example, if an EPCIS event contains a given instance-level identifier, and a subsequent EPCIS event contains the same identifier, then it is certain that the very same object participated in both events. In contrast, if both events contained class-level identifiers, then it is not certain that the same object participated in both events, because the second event could have been a different instance of the same class (i.e., a different object carrying the same class-level identifier as the first object). Class-level identifiers are typically used only when it is impractical to assign unique instance-level identifiers to each object.

Examples (non-normative): In the GS1 System, examples of instance-level identifiers include GTIN+serial, SSCC, GRAI including serial, GIAI, GSRN, and GDTI including serial. Examples of class-level identifiers include GTIN, GTIN+lot, GRAI without serial, and GDTI without serial.

7.3.3.2 EPC

An Electronic Product Code (EPC) is an instance-level identifier structure defined in the EPC Tag Data Standard [TDS1.9]. In the "what" dimension of an EPCIS event, the value of an `epc` element SHALL be a URI [RFC2396] denoting the unique instance-level identity for an object. When the unique identity is an Electronic Product Code, the list element SHALL be the "pure identity" URI for the EPC as specified in [TDS1.9], Section 6. Implementations MAY accept URI-formatted identifiers other than EPCs as the value of an `epc` element.

7.3.3.3 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.

Field	Type	Description
`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 Section 6.2. 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 summarizes 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.

EPCClass	`quantity` field	`uom` field	Meaning
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.

7.3.3.3.1 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*.

7.3.3.3.2 EPCClass Values for GTIN

When a Vocabulary Element in `EPCClass` represents the class of SGTIN EPCs denoted by a specific GTIN, it SHALL be a URI in the following form, as defined in Version 1.3 and later of the EPC Tag Data Standards:

`urn:epc:idpat:sgtin:`*`CompanyPrefix.ItemRefAndIndicator.*`

where *`CompanyPrefix`* is the GS1 Company Prefix of the GTIN (including leading zeros) and *`ItemRefAndIndicator`* consists of the indicator digit of the GTIN followed by the digits of the item reference of the GTIN.

An `EPCClass` vocabulary element in this form denotes the class of objects whose EPCs are SGTINs (`urn:epc:id:sgtin:`…) having the same `CompanyPrefix` and

966 `ItemRefAndIndicator` fields, and having any serial number whatsoever (or no serial
967 number at all).

968 7.3.3.3.3 EPCClass Values for GTIN + Batch/Lot

969 When a Vocabulary Element in `EPCClass` represents the class of SGTIN EPCs denoted by a
970 specific GTIN and batch/lot, it SHALL be a URI in the following form, as defined in [TDS1.9,
971 Section 6]:

972 `urn:epc:class:lgtin:`*`CompanyPrefix.ItemRefAndIndicator.Lot`*

973 where *`CompanyPrefix`* is the GS1 Company Prefix of the GTIN (including leading zeros),
974 *`ItemRefAndIndicator`* consists of the indicator digit of the GTIN followed by the digits of
975 the item reference of a GTIN, and *`Lot`* is the batch/lot number of the specific batch/lot.

976 An `EPCClass` vocabulary element in this form denotes the class of objects whose EPCs are
977 SGTINs (`urn:epc:id:sgtin:`…) having the same `CompanyPrefix` and
978 `ItemRefAndIndicator` fields, and belonging to the specified batch/lot, regardless of serial
979 number (if any).

980 7.3.3.4 Summary of Identifier Types (Non-Normative)

981 This section summarizes the identifiers that may be used in the "what" dimension of EPCIS
982 events. The normative specifications of identifiers are in the EPC Tag Data Standard [TDS1.9]
983 and the EPC Core Business Vocabulary [CBV1.1].

Identifier Type	Instance-Level (EPC)	Class-Level (EPCClass)	URI Prefix	Normative Reference
GTIN		✓	`urn:epc:idpat:sgtin:`	[TDS1.9, Section 8]
GTIN + batch/lot		✓	`urn:epc:class:lgtin:`	[TDS1.9, Section 6]
GTIN + serial	✓		`urn:epc:id:sgtin:`	[TDS1.9, Section 6.3.1]
SSCC	✓		`urn:epc:id:sscc:`	[TDS1.9, Section 6.3.2]
GRAI (no serial)		✓	`urn:epc:idpat:grai:`	[TDS1.9, Section 8]
GRAI (with serial)	✓		`urn:epc:id:grai:`	[TDS 1.9, Section 6.3.4]

Identifier Type	Instance-Level (EPC)	Class-Level (EPCClass)	URI Prefix	Normative Reference
GIAI	✓		`urn:epc:id:giai:`	[TDS1.9, Section 6.3.5]
GDTI (no serial)		✓	`urn:epc:idpat:gdti:`	[TDS1.9, Section 8]
GDTI (with serial)	✓		`urn:epc:id:gdti:`	[TDS1.9, Section 6.3.7]
GSRN (Supplier)	✓		`urn:epc:id:gsrn:`	[TDS1.9, Section 6.3.6]
GSRN (Provider)	✓		`urn:epc:id:gsrnp:`	[TDS1.9, Section 6.3.6]
GCN (no serial)		✓	`urn:epc:idpat:sgcn:`	[TDS1.9, Section 8]
GCN (with serial)	✓		`urn:epc:id:sgcn:`	[TDS1.9, Section 6]
CPI		✓	`urn:epc:idpat:cpi:`	[TDS1.9, Section 8]
CPI + serial	✓		`urn:epc:id:cpi:`	[TDS1.9, Section 6.3.11]
GID	✓		`urn:epc:id:gid:`	[TDS1.9, Section 6.3.8]
USDoD	✓		`urn:epc:id:usdod:`	[TDS1.9, Section 6.3.9]
ADI	✓		`urn:epc:id:adi:`	[TDS1.9, Section 6.3.10]
Non-GS1 Identifier	✓	✓	Any URI – see CBV for recommendations	[CBV1.1, Section 8.2]

984

7.3.4 The "Where" Dimension – Read Point and Business Location

This section defines four types that all relate to the notion of *location* information as used in EPCIS. Two of these types are ways of referring to "readers," or devices that sense the presence of EPC-tagged objects using RFID or other means. These are not actually considered to be "location" types at all for the purposes of EPCIS. They are included in this specification mainly to contrast them to the true location types (though some applications may want to use them as extension fields on observations, for auditing purposes).

The next two concepts are true location types, and are defined as EPCIS Vocabulary Types.

Explanation (non-normative): In the EPC context, the term location has been used to signify many different things and this has lead to confusion about the meaning and use of the term, particularly when viewed from a business perspective. This confusion stems from a number of causes:

1. In situations where EPC Readers are stationary, there's a natural tendency to equate the reader with a location, though that may not always be valid if there is more than one reader in a location;

2. There are situations where stationary Readers are placed between what business people would consider to be different locations (such as at the door between the backroom and sales floor of a retail store) and thus do not inherently determine the location without an indication of the direction in which the tagged object was travelling;

3. A single physical Reader having multiple, independently addressable antennas might be used to detect tagged objects in multiple locations as viewed by the business people;

4. Conversely, more than one Reader might be used to detect tagged objects in what business people would consider a single location;

5. With mobile Readers, a given Reader may read tagged objects in multiple locations, perhaps using "location" tags or other means to determine the specific location associated with a given read event;

6. And finally, locations of interest to one party (trading partner or application) may not be of interest to or authorized for viewing by another party, prompting interest in ways to differentiate locations.

The key to balancing these 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).

In addition, there has historically been much confusion around the difference between "location" as needed by EPCIS-level applications and reader identities. This EPCIS specification defines location as something quite distinct from reader identity. To help make this clear, the reader identity types are defined below to provide a contrast to the definitions of the true EPCIS location types. Also, reader identity types may enter into EPCIS as "observational attributes" when an application desires to retain a record of what readers played a role in an observation; e.g., for auditing purposes. (Capture and sharing of "observational attributes" would require use of extension fields not defined in this specification.)

1026 The reader/location types are as follows:

	Type	Description
Primitive Reader Types – *not* location types for EPCIS		
	`PhysicalReaderID`	This is the unique identity or name of the specific information source (e.g., a physical RFID Reader) that reports the results of an EPC read event. Physical Reader ID is further defined in [ALE1.0].
	`LogicalReaderID`	This is the identity or name given to an EPC read event information source independent of the physical device or devices that are used to perform the read event. Logical Reader ID is further defined in [ALE1.0]. There are several reasons for introducing the Logical Reader concept as outlined in [ALE1.0], including allowing physical readers to be replaced without requiring changes to EPCIS Capturing Applications, allowing multiple physical readers to be given a single name when they are always used simultaneously to cover a single location, and (conversely) allowing a single physical reader to map to multiple logical readers when a physical reader has multiple antennas used independently to cover different locations.
True Location Types		
	`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

	Type	Description
		Point, the EPCIS Capturing Application determines the Business Location based on whatever means it chooses. Conceptually, the Business Location is designed to identify "where objects are following the EPCIS event."

`ReadPointID` and `BusinessLocationID` are User Vocabularies as defined in Section 6.2. Some industries may wish to use EPCs as vocabulary elements, in which case pure identity URIs as defined in [TDS1.9] SHALL be used.

Illustration (non-normative): For example, in industries governed by GS1 General Specifications, `readPointID`, and `businessLocationID` may be SGLN-URIs [TDS1.9, Section 6.3.3], and `physicalReaderID` may be an SGTIN-URI [TDS1.9, Section 6.3.1].

But in all cases, location vocabulary elements are not *required* to be EPCs.

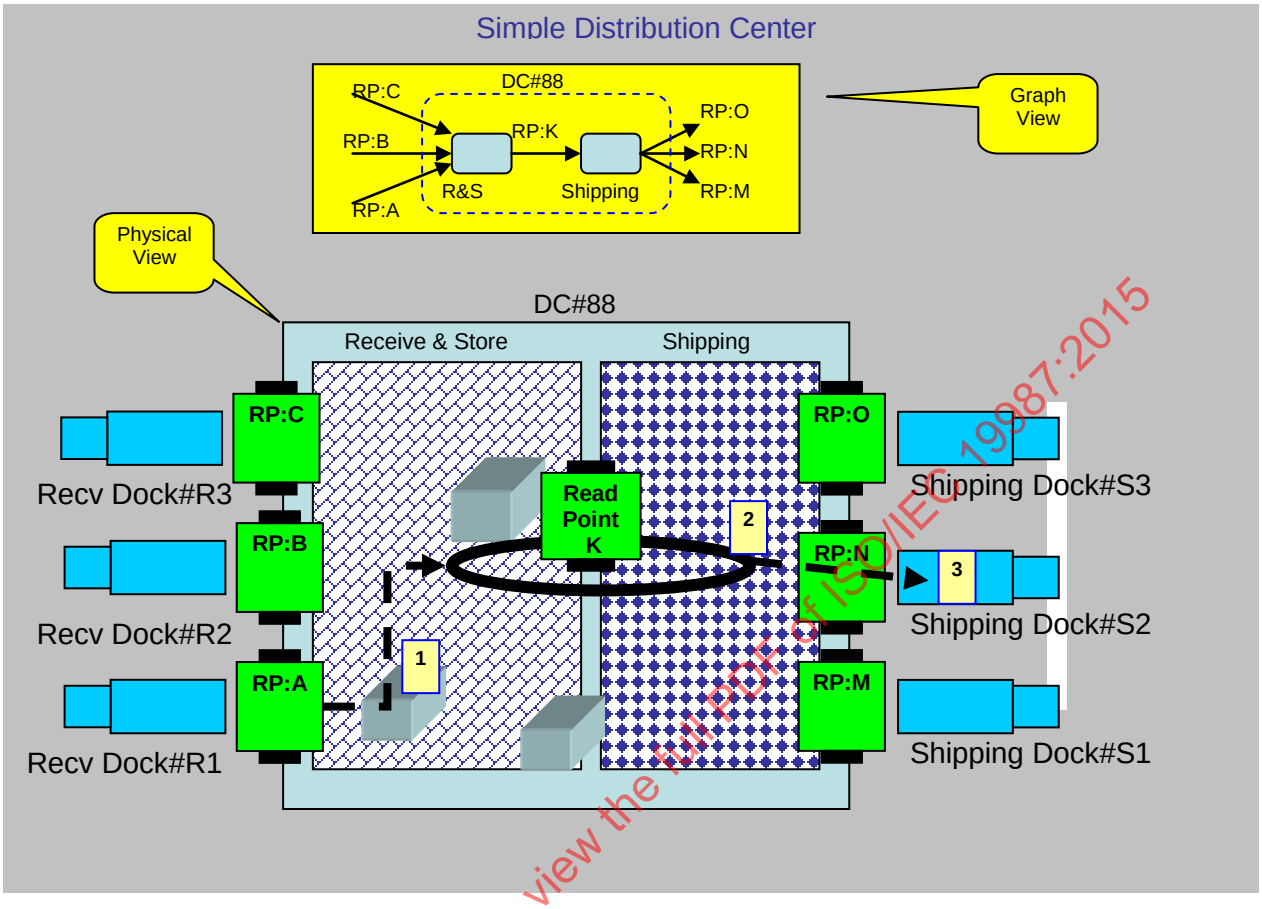
Explanation (non-normative): Allowing non-EPC URIs for locations gives organizations greater freedom to reuse existing ways of naming locations.

For all of the EPCIS Event Types defined in this Section 7.2, 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.

Explanation (non-normative): 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 organization makes a change in the way a particular business process is executed; again, not often of interest to a partner.

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

7.3.4.1 Example of the distinction between a Read Point and a Business Location (Non-Normative)



Tag	*Time*	*Read Point*	*Business Location*	*Comment*
#123	*7:00*	*"RP-DC#88-A"*	*DC#88.Receive & Store*	*Product entered DC via DockDoor#R1*
#123	*9:00*	*"RP-DC#88-K"*	*DC#88.Shipping*	*Product placed on conveyor for shipping*
#123	*9:30*	*"RP-DC#88-N"*	*[omitted]*	*Product shipped via dock door#S2*

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

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

A supply chain like the rooms-and-doors example may be represented by a graph in which each node in the graph represents a room in which objects may be found, and each arc represents a doorway that connects two rooms. Business Locations, therefore, correspond to nodes of this graph, and Read Points correspond to the arcs. If the graph were a straight, unidirectional chain, the arcs traversed by a given object could be reconstructed from knowing the nodes; that is, Read Point information would be redundant given the Business Location information. In more real-world situations, however, objects can take multiple paths and move "backwards" in the supply chain. In these real-world situations, providing Read Point information in addition to Business Location information is valuable for higher level software.

7.3.5 The "Why" Dimension

This section defines the data types used in the "Why" dimension of the event types specified in Section 7.3.5.4.

7.3.5.1 Business Step

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

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

This specification defines no Master Data Attributes for business step identifiers.

7.3.5.2 Disposition

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

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

This specification defines no Master Data Attributes for disposition identifiers.

7.3.5.3 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.

7.3.5.3.1 Business Transaction Type

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

Explanation (non-normative): 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.

7.3.5.3.2 Business Transaction ID

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

Explanation (non-normative): URIs are used to provide extensibility and a convenient way for organizations 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.

7.3.5.4 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 standardized 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`	An identifier that indicates what kind of source or destination this `Source` or `Destination` (respectively) denotes.
`source` or `destination`	`SourceDestID`	An identifier that denotes a specific source or destination.

1164

1165 The two vocabulary types `SourceDestTypeID`, and `SourceDestID` are defined in the
1166 sections below.

1167 7.3.5.4.1 Source/Destination Type

1168 `SourceDestTypeID` is a vocabulary whose elements denote a specific type of business
1169 transfer source or destination. An example is an identifier that denotes "owning party."
1170 `SourceDestTypeID` is an example of a Standard Vocabulary as defined in Section 6.2.

1171 *Explanation (non-normative): Using an extensible vocabulary for source/destination type*
1172 *identifiers allows GS1 standards to define some common terms such as "owning party" while*
1173 *allowing for industry groups and individual end-users to define their own terms. Master data*
1174 *may provide additional information.*

1175 This specification defines no Master Data Attributes for source/destination type identifiers.

1176 7.3.5.4.2 Source/Destination ID

1177 `SourceDestID` is a vocabulary whose elements denote specific sources and destinations. An
1178 example is an identifier that denotes "Acme Corporation (an owning party)." `SourceDestID`
1179 is a User Vocabulary as defined in Section 6.2.

1180 *Explanation (non-normative): URIs are used to provide extensibility and a convenient way for*
1181 *organizations to distinguish one kind of source or destination identifier from another.*

1182 7.3.6 Instance/Lot Master Data (ILMD)

1183 Instance/Lot Master Data (ILMD) is data that describes a specific instance of a physical or
1184 digital object, or a specific batch/lot of objects that are produced in batches/lots. ILMD consists
1185 of a set of descriptive attributes that provide information about one or more specific objects or
1186 lots. It is similar to ordinary Master Data, which also consists of a set of descriptive attributes
1187 that provide information about objects. But whereas Master Data attributes have the same values
1188 for a large class of objects, (e.g., for all objects having a given GTIN), the values of ILMD
1189 attributes may be different for much smaller groupings of objects (e.g., a single batch or lot), and
1190 may be different for each object (i.e., different for each instance).

1191 An example of a Master Data attribute is the weight and physical dimensions of trade items
1192 identified by a specific GTIN. These values are the same for all items sharing that GTIN. An

example of ILMD is the expiration date of a perishable trade item. Unlike Master Data, the expiration date is not the same for all trade items having the same GTIN; in principle, each may have a different expiration date depending on when it is manufactured. Other examples of ILMD include date of manufacture, place of manufacture, weight and other physical dimensions of a variable-measure trade item, harvest information for farm products, and so on.

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

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

ILMD for a specific object is defined when the object comes into existence. Therefore, ILMD may only be included in `ObjectEvent`s with action `ADD` (Section 7.4.2), and in `TransformationEvent`s (Section 7.4.6). In the case of a `TransformationEvent`, ILMD applies to the outputs of the transformation, not to the inputs.

The structure of ILMD defined in this EPCIS standard consists of a set of named attributes, with values of any type. In the XML binding (Section 9.5), the XML schema provides for an unbounded list of XML elements having any element name and content. Other documents layered on top of EPCIS may define specific ILMD data elements; see Section 6.3. In this way, ILMD is similar to event-level EPCIS extensions, but is separate in order to emphasize 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`	`Time`	The date and time at which the EPCIS Capturing Applications asserts the event occurred.

Field	Type	Description
`recordTime`	`Time`	(Optional) The date and time at which this event was recorded by an EPCIS Repository. This field SHALL be ignored when an event is presented to the EPCIS Capture Interface, and SHALL be present when an event is retrieved through the EPCIS Query Interfaces. The `recordTime` does not describe anything about the real-world event, but is rather a bookkeeping mechanism that plays a role in the interpretation of standing queries as specified in Section 8.2.5.2.
`eventTimeZoneOffset`	`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).

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

The purpose of `eventTimeZoneOffset` is to provide additional business context about the event, namely to identify what time zone offset was in effect at the time and place the event was captured. This information may be useful, for example, to determine whether an event took place during business hours, to present the event to a human in a format consistent with local time, and so on. The local time zone offset information is not *necessarily available from `eventTime`, because there is no requirement that the time zone specifier in the XML representation of `eventTime` be the local time zone offset where the event was captured. For example, an event taking place at 8:00am US Eastern Standard Time could have an XML `eventTime` field that is written `08:00-05:00` (using US Eastern Standard Time), or `13:00Z` (using UTC), or even `07:00-06:00` (using US Central Standard Time). Moreover, XML processors are not required by [XSD2] to retain and present to applications the time zone specifier that was part of the `xsd:dateTime` 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.*

7.4.2 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.

Action value	Meaning
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).

1262

1263 Fields:

Field	Type	Description
eventTime recordTime eventTimeZoneOffset	(Inherited from EPCISEvent; see Section 7.4.1)	
epcList	List<EPC>	(Optional) An unordered list of one or more EPCs naming specific objects to which the event pertained. See Section 7.3.3.2. An ObjectEvent SHALL contain either a non-empty epcList, a non-empty quantityList, or both.
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.
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.

Field	Type	Description
`disposition`	`DispositionID`	(Optional) The business condition of the objects associated with the EPCs, presumed to hold true until contradicted by a subsequent event.
`readPoint`	`ReadPointID`	(Optional) The read point at which the event took place.
`bizLocation`	`BusinessLocationID`	(Optional) The business location where the objects associated with the EPCs may be found, until contradicted by a subsequent event.
`bizTransactionList`	Unordered list of zero or more `BusinessTransaction` instances	(Optional) An unordered list of business transactions that define the context of this event.
`sourceList`	`List<Source>`	(Optional) An unordered list of `Source` elements (Section 7.3.5.4) that provide context about the originating endpoint of a business transfer of which this event is a part.
`destinationList`	`List<Destination>`	(Optional) An unordered list of `Destination` elements (Section 7.3.5.4) that provide context about the terminating endpoint of a business transfer of which this event is a part.
`ilmd`	`ILMD`	(Optional) Instance/Lot Master Data (Section 7.3.6) that describes the objects created during this event. An `ObjectEvent` SHALL NOT contain `ilmd` if `action` is `OBSERVE` or `DELETE`.

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

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`.

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 associated with identified by `epcList` and `quantityList` is unchanged.
- (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`.

Explanation (non-normative): 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`.

Explanation (non-normative): 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.

Explanation (non-normative): 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 `AggregateEvent` 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 Section 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 Section 7.4.1)	

`parentID`	`URI`	(Optional when `action` is `OBSERVE`, required otherwise.) The identifier of the parent of the association. When the parent identifier is an EPC, this field SHALL contain the "pure identity" URI for the EPC as specified in [TDS1.9], Section 7.
`childEPCs`	`List<EPC>`	(Optional) An unordered list of the EPCs of contained objects identified by instance-level identifiers. See Section 7.3.3.2. 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.

Field	Type	Description
childQuantityList	List<QuantityElement>	(Optional) An unordered list of one or more QuantityElements identifying (at the class level) contained objects. See Section 7.3.3.2. 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.
readPoint	ReadPointID	(Optional) The read point at which the event took place.

Field	Type	Description
`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.
`sourceList`	`List<Source>`	(Optional) An unordered list of `Source` elements (Section 7.3.5.4) that provide context about the originating endpoint of a business transfer of which this event is a part.
`destinationList`	`List<Destination>`	(Optional) An unordered list of `Destination` elements (Section 7.3.5.4) that provide context about the terminating endpoint of a business transfer of which this event is a part.

Note that in the XML binding (Section 9.3), `childQuantityList`, `sourceList`, and `destinationList` appear in the standard extension area, to maintain forward-compatibility with EPCIS 1.0.

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 `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).

Explanation (non-normative): 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 a AggregationEvent with a non-empty* `bizTransactionList` *does not cause a TransactionEvent to be returned from a query.*

Note (non-normative): 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 QuantityEvent (subclass of EPCISEvent) – DEPRECATED

A `QuantityEvent` captures an event that takes place with respect to a specified quantity of an object class. This Event Type may be used, for example, to report inventory levels of a product.

As of EPCIS 1.1, the `QuantityEvent` is deprecated. Applications should instead use an `ObjectEvent` containing one or more `QuantityListElement`s. A `QuantityEvent` is equivalent to an `ObjectEvent` containing an empty `EPCList` and a single `QuantityListElement` containing a `quantity` and without a `uom`.

Field	Type	Description

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

`epcClass`	`EPCClass`	The identifier specifying the object class to which the event pertains.
`quantity`	`Int`	The quantity of object within the class described by this event.
`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.
`readPoint`	`ReadPointID`	(Optional) The read point at which the event took place.

Field	Type	Description
`bizLocation`	`BusinessLocationID`	(Optional) The business location where the objects 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.

Note that because an EPCClass always denotes a specific packaging unit (e.g., a 12-item case), there is no need for an explicit "unit of measure" field. The unit of measure is always the object class denoted by `epcClass` as defined in Master Data for that object class.

Retrospective semantics:

- An event described by `bizStep` (and any other fields) took place with respect to `quantity` objects of EPC class `epcClass` at `eventTime` at location `readPoint`.
- (If a non-empty `bizTransactionList` is specified) This event took place within the context of the business transactions enumerated in `bizTransactionList`.

Prospective semantics: .

- (If `disposition` is specified) The business condition of the objects is as described by `disposition`.
- (If `disposition` is omitted) The business condition of the objects is unchanged.
- (If `bizLocation` is specified) The objects are at business location `bizLocation`.
- (If `bizLocation` is omitted) The business location of the objects is unknown.

7.4.5 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.

1461

1462 A `TransactionEvent` has the following fields:

Field	Type	Description

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

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

Field	Type	Description
`parentID`	`URI`	(Optional) The identifier of the parent of the objects given in `epcList` and `quantityList`. When the parent identifier is an EPC, this field SHALL contain the "pure identity" URI for the EPC as specified in [TDS1.9], Section 7. See also the note following the table.
`epcList`	`List<EPC>`	(Optional) An unordered list of the EPCs of the objects identified by instance-level identifiers associated with the business transaction. See Section 7.3.3.2. A TransactionEvent SHALL contain either a non-empty `epcList`, a non-empty `quantityList`, or both, except that both `epcList` and `quantityList` MAY be empty if `action` is `DELETE`, indicating that all the objects are disassociated from the business transaction(s).

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

Field	Type	Description
`sourceList`	`List<Source>`	(Optional) An unordered list of `Source` elements (Section 7.3.5.4) that provide context about the originating endpoint of a business transfer of which this event is a part.
`destinationList`	`List<Destination>`	(Optional) An unordered list of `Destination` elements (Section 7.3.5.4) that provide context about the terminating endpoint of a business transfer of which this event is a part.

Note that in the XML binding (Section 9.3), `quantityList`, `sourceList`, and `destinationList` appear in the standard extension area, to maintain forward-compatibility with EPCIS 1.0.

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

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

Retrospective semantics:

- An event described by `bizStep` (and any other fields) took place involving the business transactions enumerated in `bizTransactionList`, the objects in `epcList` and `quantityList`, and containing entity `parentID` (if specified), at `eventTime` and location `readPoint`.

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

Prospective semantics:

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

7.4.6 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 `TransfomationID` may be included in two or more `TransformationEvents` to link them together. When events share an identical `TransformationID`, the meaning is that the inputs to *any* of those events may have contributed in some way to each of the outputs in *any* of those same events.

Fields:

Field	Type	Description

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

`inputEPCList`	`List<EPC>`	(Optional) An unordered list of one or more EPCs identifying (at the instance level) objects that were inputs to the transformation. See Section 7.3.3.2. See below for constraints on when `inputEPCList` may be omitted.
`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. See below for constraints on when `inputQuantityList` may be omitted.

Field	Type	Description
`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 Section 7.3.3.2. See below for constraints on when `outputEPCList` may be omitted.
`outputQuantityList`	`List<QuantityElement>`	(Optional) An unordered list of one or more QuantityElements identifying (at the class level) objects that were outputs from the transformation. See below for constraints on when `outputQuantityList` may be omitted.

Field	Type	Description
`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.
`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.

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 (Section 7.3.5.4) that provide context about the originating endpoint of a business transfer of which this event is a part.
`destinationList`	`List<Destination>`	(Optional) An unordered list of `Destination` elements (Section 7.3.5.4) that provide context about the terminating endpoint of a business transfer of which this event is a part.
`ilmd`	`ILMD`	(Optional) Instance/Lot Master Data (Section 7.3.6) that describes the output objects created during this event.

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`.

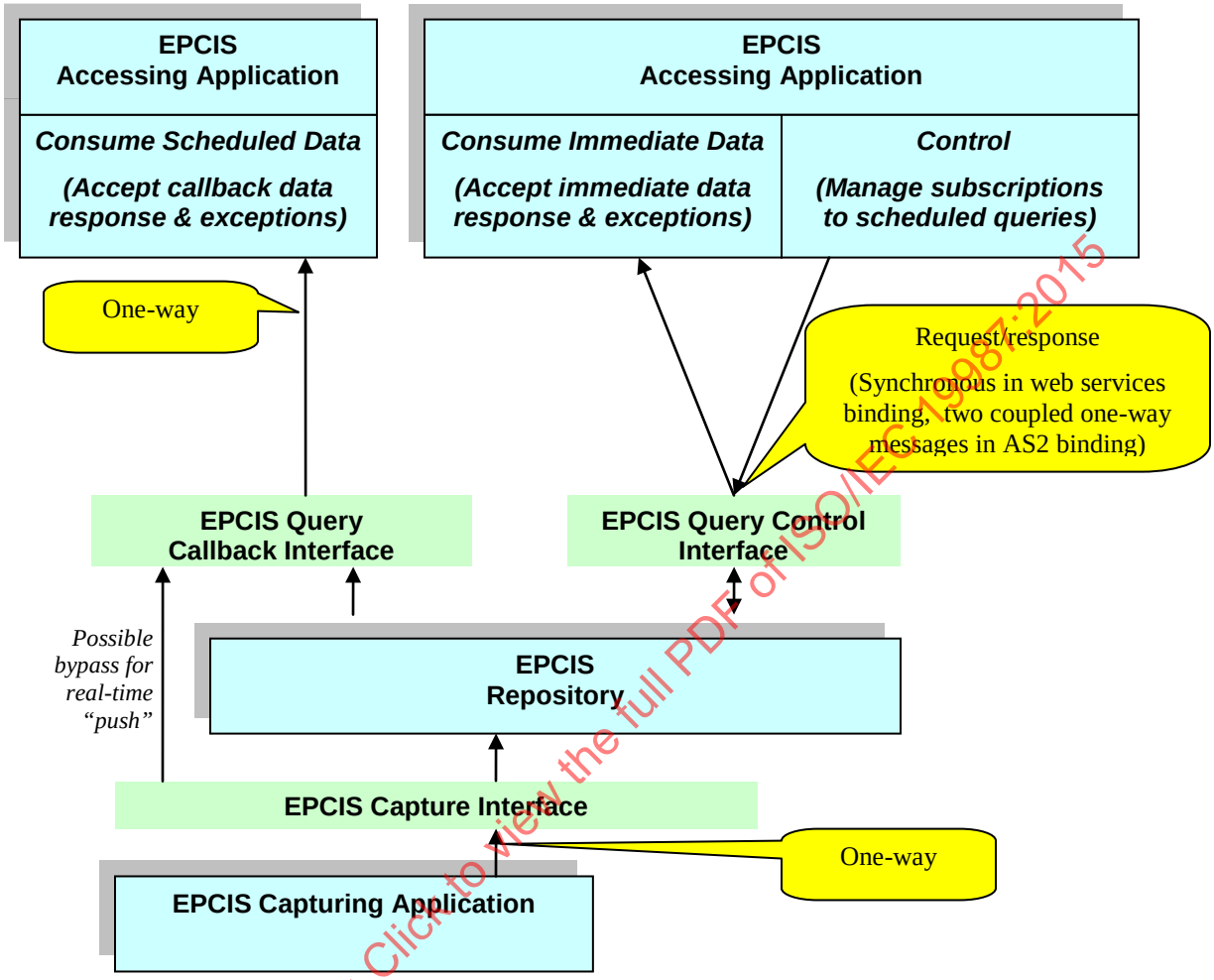
Prospective semantics:

- 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 `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`.

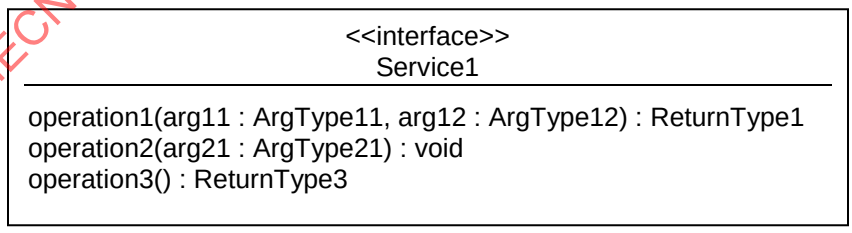
8 Service Layer

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

between these interfaces, expanding upon the diagram in Section 2 (this diagram is non-normative):



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.

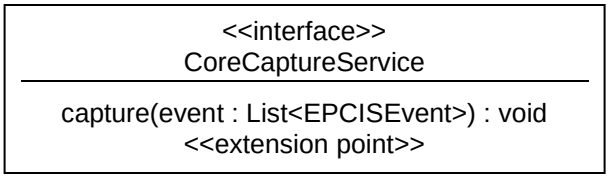
8.1.1 Authentication and Authorization

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 authorization decisions as described below. Moreover, an implementation MAY record the client identity with the captured data, for use in subsequent authorization decisions by the system implementing the EPCIS Query Interfaces, as described in Section 8.2.2.

Because of the simplicity of the EPCIS Capture Interface, the authorization 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.

Explanation (non-normative): 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 organizational boundaries. The bindings that do not offer authentication are expected to be used only within a single organization 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.

Explanation (non-normative): "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 (serialized 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. *Explanation (non-normative): this treatment of `recordTime` is necessary in order for standing queries to be processed properly. See Section 8.2.5.2.*

Return value:

(none)

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 authorization decisions as described in the next section.

Explanation (non-normative): 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 organizational boundaries. The bindings that do not offer authentication are expected to be used only within a single organization in situations where authentication is not required to meet internal security requirements.

8.2.2 Authorization

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 organization than the operator of an EPCIS service, but may also arise in intra-enterprise scenarios.

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

- The service MAY refuse to honour the request altogether, by responding with a `SecurityException` as defined below.
- The service MAY respond with less data than requested. For example, if a client presents a query requesting all `ObjectEvent` instances within a specified time interval, the service knows of 100 matching events, the service may choose to respond with fewer than 100 events (e.g., returning only those events whose EPCs are SGTINs with a company prefix known to be assigned to the client).
- The service MAY respond with coarser grained information. In particular, when the response to a query includes a location type (as defined in Section 7.3.4), the service may substitute an aggregate location in place of a primitive location.

- The service MAY hide information. For example, if a client presents a query requesting `ObjectEvent` instances, the service may choose to delete the `bizTransactionList` fields in its response. The information returned, however, SHALL be well-formed EPCIS events consistent with this specification and industry guidelines. In addition, if hiding information would otherwise result in ambiguous, or misleading information, then the entire event SHOULD be withheld. This applies whether the original information was captured through the EPCIS Capture Interface or provided by some other means. For example, given an AggregationEvent with action equal to ADD, an attempt to hide the `parentID` field would result in a non-well-formed event, because `parentID` is required when the action is ADD; in this instance, therefore, the entire event would have to be withheld.
- The service MAY limit the scope of the query to data that was originally captured by a particular client identity. This allows a single EPCIS service to be "partitioned" for use by groups of unrelated users whose data should be kept separate.

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

Explanation (non-normative): 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 authorization decisions are taken. Particular implementations of EPCIS may have arbitrarily complex business rules for authorization. That said, the EPCIS specification may contain standard data that is needed for authorization, 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).

Roadmap (non-normative): 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 (Section 8.2.3), this may present performance problems for service implementations.

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

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

8.2.5 Query Framework (EPCIS Query Control Interface)

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

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

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

The specification of these methods is as follows:

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

Method	Description
`getStandardVersion`	Returns a string that identifies what version of the 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 to which it complies. To indicate compliance with this Version 1.1 of the EPCIS specification, the implementation SHALL return the string `1.1`.
`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.

1793

1794 This framework applies regardless of the content of a query. The detailed contents of a query,
1795 and the results as returned from `poll` or delivered to a subscriber via the Query Callback
1796 Interface, are defined in later sections of this document. This structure is designed to facilitate
1797 extensibility, as new types of queries may be specified and fit into this general framework.

1798 An implementation MAY restrict the behaviour of any method according to authorization
1799 decisions based on the authenticated client identity of the client making the request. For example,
1800 an implementation may limit the IDs returned by `getSubscriptionIDs` and recognized by
1801 `unsubscribe` to just those subscribers that were previously subscribed by the same client
1802 identity. This allows a single EPCIS service to be "partitioned" for use by groups of unrelated
1803 users whose data should be kept separate.

1804 If a pre-defined query defines named parameters, values for those parameters may be supplied
1805 when the query is subsequently referred to using `poll` or `subscribe`. A `QueryParams`
1806 instance is simply a set of name/value pairs, where the names correspond to parameter names
1807 defined by the query, and the values are the specific values to be used for that invocation of
1808 (`poll`) or subscription to (`subscribe`) the query. If a `QueryParams` instance includes a
1809 name/value pair where the value is empty, it SHALL be interpreted as though that query
1810 parameter were omitted altogether.

1811 The `poll` or `subscribe` method SHALL raise a `QueryParameterException` under any
1812 of the following circumstances:

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

8.2.5.1 Subscription Controls

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

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

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

Argument	Type	Description
`schedule`	`QuerySchedule`	(Optional) Defines the periodic schedule on which the query is to be executed. See Section 8.2.5.3. Exactly one of `schedule` or `trigger` is required; if both are specified or both are omitted, the implementation SHALL raise a `SubscriptionControls-Exception`..

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 `SubscriptionControls-Exception`..
`initialRecordTime`	`Time`	(Optional) Specifies a time used to constrain what events are considered when processing the query when it is executed for the first time. See Section 8.2.5.2. If omitted, defaults to the time at which the subscription is created.
`reportIfEmpty`	`boolean`	If true, a `QueryResults` instance is always sent to the subscriber when the query is executed. If false, a `QueryResults` instance is sent to the subscriber only when the results are non-empty.

8.2.5.2 Automatic Limitation Based On Event Record Time

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

For this reason, an EPCIS service SHALL constrain the scope of each query execution for a subscribed query in the following manner. The first time the query is executed for a given subscription, the only events considered are those whose `recordTime` field is greater than or equal to `initialRecordTime` specified when the subscription was created. For each execution of the query following the first, the only events considered are those whose `recordTime` field is greater than or equal to the time when the query was last executed. It is implementation dependent as to the extent that failure to deliver query results to the subscriber affects this calculation; implementations SHOULD make best efforts to insure reliable delivery

of query results so that a subscriber does not miss any data. The query or query parameters may specify additional constraints upon record time; these are applied after restricting the universe of events as described above.

Explanation (non-normative): 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 `Elements` 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, authorization, 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.

Explanation (non-normative): 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.

Argument	Type	Description
`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.*

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.

Field	Type	Description
`resultsBody`	`QueryResultsBody`	The information returned as the result of a query. The exact type of this field depends on which query is executed. Each of the predefined queries in Section 8.2.7 specifies the corresponding type for this field.

8.2.6 Error Conditions

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

Exception Name	Meaning
`SecurityException`	The operation was not permitted due to an access control violation or other security concern. This includes the case where the service wishes to deny authorization 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 1.0) The specified query name already exists.
`QueryValidationException`	(Not implemented in EPCIS 1.0) The specified query is invalid; e.g., it contains a syntax error.
`QueryParameterException`	One or more query parameters are invalid, including any of the following situations: • the parameter name is not a recognized 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.

Exception Name	Meaning
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 recognized 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 recognized 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.

1966

1967 The exceptions that may be thrown by each method of the EPCIS Query Control Interface are
1968 indicated in the table below:

EPCIS Method	Exceptions

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

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

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

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

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

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

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

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

addition to the reason string, the query name and the `subscriptionID` as specified in the `subscribe` call that created the standing query.

8.2.7 Predefined Queries for EPCIS

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

The parameters for each predefined query and what results it returns are specified in this section. An implementation of EPCIS is free to use any internal representation for data it wishes, and implement these predefined queries using any database or query technology it chooses, so long as the results seen by a client are consistent with this specification.

8.2.7.1 SimpleEventQuery

This query is invoked by specifying the string `SimpleEventQuery` as the `queryName` argument to `poll` or `subscribe`. The result is a `QueryResults` instance whose body contains a (possibly empty) list of `EPCISEvent` instances. Unless constrained by the `eventType` parameter, each element of the result list could be of any event type; i.e., `ObjectEvent`, `AggregationEvent`, `QuantityEvent`, `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 authorization (Section 8.2.2), the inclusion of the `recordTime` field, and any necessary conversions to and from an abstract internal representation. For any event field defined to hold an unordered list, however, an EPCIS implementation 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
`eventType`	List of String	If specified, the result will only include events whose type matches one of the types specified in the parameter value. Each element of the parameter value may be one of the following strings: `ObjectEvent`, `AggregationEvent`, `QuantityEvent`, `TransactionEvent`, or `TransformationEvent`. An element of the parameter value may also be the name of an extension event type. If omitted, all event types will be considered for inclusion in the result.
`GE_eventTime`	Time	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`	Time	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`	Time	If provided, only events with `recordTime` greater than or equal to the specified value will be returned. The automatic limitation based on event record time (Section 8.2.5.2) may implicitly provide a constraint similar to this parameter. If omitted, events are included regardless of their `recordTime`, other than automatic limitation based on event record time (Section 8.2.5.2).

Parameter Name	Parameter Value Type	Meaning
`LT_recordTime`	Time	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 elements 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_readPoint`	List of String	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 values. 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.

Parameter Name	Parameter Value Type	Meaning
`WD_readPoint`	List of String	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 values, or is a direct or indirect descendant of one of the specified values. The meaning of "direct or indirect descendant" is specified by master data, as described in Section 6.5. (`WD` is an abbreviation for "with descendants.") If this parameter and `EQ_readPoint` are both omitted, events are returned regardless of the value of the `readPoint` field or whether the `readPoint` field exists at all.
`EQ_bizLocation`	List of String	Like the `EQ_readPoint` parameter, but for the `bizLocation` field.
`WD_bizLocation`	List of String	Like the `WD_readPoint` parameter, but for the `bizLocation` field.
`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.

Parameter Name	Parameter Value Type	Meaning
`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, `TransformationEvent`s or extension event type that extend `TransformationEvent`); and where (b) the `transformationID` field is equal to one of the values specified in this parameter.
`MATCH_epc`	List of String	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 EPC patterns or URIs specified in this parameter, where the meaning of "matches" is as specified in Section 8.2.7.1.1. If this parameter is omitted, events are included regardless of their `epcList` or `childEPCs` field or whether the `epcList` or `childEPCs` field exists.

Parameter Name	Parameter Value Type	Meaning
`MATCH_parentID`	List of String	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 Section 8.2.7.1.1.
`MATCH_inputEPC`	List of String	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 EPC patterns or URIs specified in this parameter. The meaning of "matches" is as specified in Section 8.2.7.1.1. If this parameter is omitted, events are included regardless of their inputEPCList field or whether the inputEPCList field exists.
`MATCH_outputEPC`	List of String	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 EPC patterns or URIs specified in this parameter. The meaning of "matches" is as specified in Section 8.2.7.1.1. If this parameter is omitted, events are included regardless of their outputEPCList field or whether the outputEPCList field exists.

Parameter Name	Parameter Value Type	Meaning
`MATCH_anyEPC`	List of String	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 the EPC patterns or URIs specified in this parameter. The meaning of "matches" is as specified in Section 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 `QuantityEvent`s whose `epcClass` field matches one of the EPC patterns or URIs specified in this parameter. The meaning of "matches" is as specified in Section 8.2.7.1.1.
`MATCH_inputEPCClass`	List of String	If this parameter is specified, the result will only include events that (a) have an `inputQuantityList` field (that is, `TransformationEvent` or extension event types that extend it); and where (b) one of the EPC classes listed in the `inputQuantityList` field (depending on event type) matches one of the EPC patterns or URIs specified in this parameter. The meaning of "matches" is as specified in Section 8.2.7.1.1.

Parameter Name	Parameter Value Type	Meaning
`MATCH_outputEPCClass`	List of String	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 EPC patterns or URIs specified in this parameter. The meaning of "matches" is as specified in Section 8.2.7.1.1.
`MATCH_anyEPCClass`	List of String	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 `QuantityEvent`s whose `epcClass` field matches one of the EPC patterns or URIs specified in this parameter. The meaning of "matches" is as specified in Section 8.2.7.1.1.
`EQ_quantity`	Int	(DEPCRECATED in EPCIS 1.1) If this parameter is specified, the result will only include events that (a) have a `quantity` field (that is, `QuantityEvent`s or extension event type that extend `QuantityEvent`); and where (b) the `quantity` field is equal to the specified parameter.
`GT_quantity`	Int	(DEPCRECATED in EPCIS 1.1) Like `EQ_quantity`, but includes events whose `quantity` field is greater than the specified parameter.

Parameter Name	Parameter Value Type	Meaning
`GE_quantity`	Int	(DEPCRECATED in EPCIS 1.1) Like `EQ_quantity`, but includes events whose `quantity` field is greater than or equal to the specified parameter.
`LT_quantity`	Int	(DEPCRECATED in EPCIS 1.1) Like `EQ_quantity`, but includes events whose `quantity` field is less than the specified parameter.
`LE_quantity`	Int	(DEPCRECATED in EPCIS 1.1) Like `EQ_quantity`, but includes events whose `quantity` field 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 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 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 `EQ_`, the namespace URI for the extension field, a pound sign (#), and the name of the extension field.
`EQ_`*`fieldname`*	Int Float Time	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`*.

Parameter Name	Parameter Value Type	Meaning
GT_*fieldname*	Int Float Time	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 Time	Analogous to GT_*fieldname*
EQ_ILMD_*fieldname*	List of String	Analogous to EQ_*fieldname*, but matches events whose ILMD area (Section 7.3.6) contains a field having the specified *fieldname* whose value matches one of the specified values.
EQ_ILMD_*fieldname* GT_ILMD_*fieldname* GE_ILMD_*fieldname* LT_ILMD_*fieldname* LE_ILMD_*fieldname*	Int Float Time	Analogous to EQ_*fieldname*, GT_*fieldname*, GE_*fieldname*, GE_*fieldname*, LT_*fieldname*, and LE_*fieldname*, respectively, but matches events whose ILMD area (Section 7.3.6) contains a field having the specified *fieldname* whose integer, float, or time value matches the specified value according to the specified relational operator.
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.

Parameter Name	Parameter Value Type	Meaning
EXISTS_ ILMD_*fieldname*	Void	Like EXISTS_*fieldname* as described above, but events that have a non-empty field named *fieldname* in the ILMD area (Section 7.3.6). *Fieldname* is constructed as for EQ_ILMD_*fieldname*. Note that the value for this query parameter is ignored.
HASATTR_ *fieldname*	List of String	This is not a single parameter, but a family of parameters. If a parameter of this form is specified, the result will only include events that (a) have a field named *fieldname* whose type is a vocabulary type; and (b) where the value of that field is a vocabulary element for which master data is available; and (c) the master data has a non-null attribute whose name matches one of the values specified in this parameter. *Fieldname* is the fully qualified name of a field. For a standard field, this is simply the field name; e.g., bizLocation. For an extension field, the name of an extension field is an XML qname; that is, a pair consisting of an XML namespace URI and a name. The name of the corresponding query parameter is constructed by concatenating the following: the string HASATTR_, the namespace URI for the extension field, a pound sign (#), and the name of the extension field.

Parameter Name	Parameter Value Type	Meaning
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.*

Parameter Name	Parameter Value Type	Meaning
`orderBy`	String	If specified, names a single field that will be used to order the results. The `orderDirection` field specifies whether the ordering is in ascending sequence or descending sequence. Events included in the result that lack the specified field altogether may occur in any position within the result event list. The value of this parameter SHALL be one of: `eventTime`, `recordTime`, or the fully qualified name of an extension field whose type is Int, Float, Time, or String. A fully qualified fieldname is constructed as for the `EQ_`*`fieldname`* parameter. In the case of a field of type String, the ordering SHOULD be in lexicographic order based on the Unicode encoding of the strings, or in some other collating sequence appropriate to the locale. If omitted, no order is specified. The implementation MAY order the results in any order it chooses, and that order MAY differ even when the same query is executed twice on the same data. (In EPCIS 1.0, the value `quantity` was also permitted, but its use is deprecated in EPCIS 1.1.)
`orderDirection`	String	If specified and `orderBy` is also specified, specifies whether the results are ordered in ascending or descending sequence according to the key specified by `orderBy`. The value of this parameter must be one of `ASC` (for ascending order) or `DESC` (for descending order); if not, the implementation SHALL raise a `QueryParameterException`. If omitted, defaults to `DESC`.
`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.

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

Parameter Name	Parameter Value Type	Meaning
`maxEventCount`	Int	If specified, at most this many events will be included in the query result. If the query would otherwise return more than this number of events, a `QueryTooLargeException` SHALL be raised instead of a normal query result. This parameter and e`ventCountLimit` are mutually exclusive; if both are specified, a `QueryParameterException` SHALL be raised. If this parameter is omitted, any number of events may be included in the query result. Note, however, that the EPCIS implementation is free to raise a `QueryTooLargeException` regardless of the setting of this parameter (see Section 8.2.3).

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 [TDS1.9], 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, Section 8]. 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 [TDS1.9, Section 8].

Explanation (non-normative): 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 Section 7.3.3.3). 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

This query is invoked by specifying the string `SimpleMasterDataQuery` as the `queryName` argument to `poll`. The result is a `QueryResults` instance whose body contains a (possibly empty) list of vocabulary elements together with selected attributes.

The `SimpleMasterDataQuery` SHALL be available via `poll` but not via `subscribe`; that is, an implementation SHALL raise `SubscribeNotPermittedException` when `SimpleMasterDataQuery` is specified as the `queryName` argument to `subscribe`.

The parameters for this query are as follows:

Parameter Name	Parameter Value Type	Required	Meaning
`vocabularyName`	List of String	No	If specified, only vocabulary elements drawn from one of the specified vocabularies will be included in the results. Each element of the specified list is the formal URI name for a vocabulary; e.g., one of the URIs specified in the table at the end of Section 7.2. If omitted, all vocabularies are considered.

Parameter Name	Parameter Value Type	Required	Meaning
`includeAttributes`	Boolean	Yes	If true, the results will include attribute names and values for matching vocabulary elements. If false, attribute names and values will not be included in the result.
`includeChildren`	Boolean	Yes	If true, the results will include the children list for matching vocabulary elements. If false, children lists will not be included in the result.
`attributeNames`	List of String	No	If specified, only those attributes whose names match one of the specified names will be included in the results. If omitted, all attributes for each matching vocabulary element will be included. (To obtain a list of vocabulary element names with no attributes, specify false for `includeAttributes`.) The value of this parameter SHALL be ignored if `includeAttributes` is false. Note that this parameter does not affect which vocabulary elements are included in the result; it only limits which attributes will be included with each vocabulary element.
`EQ_name`	List of String	No	If specified, the result will only include vocabulary elements whose names are equal to one of the specified values. If this parameter and `WD_name` are both omitted, vocabulary elements are included regardless of their names.

Parameter Name	Parameter Value Type	Required	Meaning
`WD_name`	List of String	No	If specified, the result will only include vocabulary elements that either match one of the specified names, or are direct or indirect descendants of a vocabulary element that matches one of the specified names. The meaning of "direct or indirect descendant" is described in Section 6.5. (`WD` is an abbreviation for "with descendants.") If this parameter and `EQ_name` are both omitted, vocabulary elements are included regardless of their names.
`HASATTR`	List of String	No	If specified, the result will only include vocabulary elements that have a non-null attribute whose name matches one of the values specified in this parameter.
`EQATTR_`*`attrname`*	List of String	No	This is not a single parameter, but a family of parameters. If a parameter of this form is specified, the result will only include vocabulary elements that have a non-null attribute named *`attrname`*, and where the value of that attribute matches one of the values specified in this parameter.

Parameter Name	Parameter Value Type	Required	Meaning
`maxElementCount`	Int	No	If specified, at most this many vocabulary elements will be included in the query result. If the query would otherwise return more than this number of vocabulary elements, a `QueryTooLargeException` SHALL be raised instead of a normal query result. If this parameter is omitted, any number of vocabulary elements may be included in the query result. Note, however, that the EPCIS implementation is free to raise a `QueryTooLargeException` regardless of the setting of this parameter (see Section 8.2.3).

As the descriptions above suggest, if multiple parameters are specified a vocabulary element 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 `WD_name` and `HASATTR` parameters, then a vocabulary element must be a descendant of the specified element AND possess one of the specified attributes in order to be included in the result.

On the other hand, for those parameters whose value is a list, a vocabulary element 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 `EQATTR_sample` parameter is a two element list (“`s1`”, “`s2`”), then a vocabulary element is included if it has a `sample` attribute whose value is equal to `s1` OR equal to `s2`.

As another example, if the value of the `EQ_name` parameter is a two element list (“`ve1`”, “`ve2`”) and the `EQATTR_sample` parameter is a two element list (“`s1`”, “`s2`”), then the effect is to include events satisfying the following predicate:

((`name` = “ve1” `OR` `name` = “ve2”)
`AND` (`sample` = “s1” `OR` `sample` = “s2”))

where `name` informally refers to the name of the vocabulary element and `sample` informally refers to the value of the `sample` attribute.

8.2.8 Query Callback Interface

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

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

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

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

9 XML Bindings for Data Definition Modules

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

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

9.1 Extensibility Mechanism

The XML schema in this section implements the `<<extension point>>` given in the UML of Section 6 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 standardized 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 1.1 schema will conform both to the GS1 Version 1.0 schema and to the GS1 Version 1.1 schema, but validation of the new features will only be available using the Version 1.1 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.)

Explanation (non-normative): 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.

Explanation (non-normative): 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 1.1 XML schema for core events (Section 9.5), this results in the following:

- Event types defined in EPCIS 1.0 appear within the `<EventList>` element.
- Event types defined in EPCIS 1.1 (i.e., TransformationEvent) each appear within an `<extension>` element within the `<EventList>` element.
- For event types defined in EPCIS 1.0, new fields added in EPCIS 1.1 appear within the `<extension>` element that follows the EPCIS 1.0 fields. If additional fields are added in a future version of EPCIS, they will appear within a second `<extension>` element that is nested within the first `<extension>` element, following the EPCIS 1.1 fields.
- For event types 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.
- 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 Section 9.6 for 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 a required 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`

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

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

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

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

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

9.3 EPCglobal Base Schema

The XML binding for the Core Event Types data definition module, as well as other XML bindings in this specification, make reference to the EPCglobal Base Schema. This schema is reproduced below.

```
<xsd:schema targetNamespace="urn:epcglobal:xsd:1"
            xmlns:epcglobal="urn:epcglobal:xsd:1"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            elementFormDefault="unqualified"
            attributeFormDefault="unqualified"
            version="1.0">
  <xsd:annotation>
    <xsd:documentation>
      <epcglobal:copyright>Copyright (C) 2004 Epcglobal Inc., All Rights
Reserved.</epcglobal:copyright>
      <epcglobal:disclaimer>EPCglobal Inc., its members, officers, directors, employees, or
agents shall not be liable for any injury, loss, damages, financial or otherwise, arising from,
related to, or caused by the use of this document.  The use of said document shall constitute
your express consent to the foregoing exculpation.</epcglobal:disclaimer>
      <epcglobal:specification>EPCglobal common components Version 1.0</epcglobal:specification>
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType name="Document" abstract="true">
    <xsd:annotation>
      <xsd:documentation xml:lang="en">
         EPCglobal document properties for all messages.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:attribute name="schemaVersion" type="xsd:decimal" use="required">
      <xsd:annotation>
        <xsd:documentation xml:lang="en">
           The version of the schema corresponding to which the instance conforms.
        </xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>
    <xsd:attribute name="creationDate" type="xsd:dateTime" use="required">
      <xsd:annotation>
        <xsd:documentation xml:lang="en">
           The date the message was created. Used for auditing and logging.
        </xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>
  </xsd:complexType>
  <xsd:complexType name="EPC">
    <xsd:annotation>
      <xsd:documentation xml:lang="en">
         EPC represents the Electronic Product Code.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:simpleContent>
      <xsd:extension base="xsd:string"/>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:schema>
```


9.4 Additional Information in Location Fields

The XML binding for the Core Event Types data definition module includes a facility for the inclusion of additional, industry-specific information in the `readPoint` and `bizLocation` fields of all event types. An industry group or other set of cooperating trading partners MAY include additional subelements within the `readPoint` or `bizLocation` fields, following the required `id` subelement. This facility MAY be used to communicate master data for location identifiers, or for any other purpose.

In all cases, however, the `id` subelement SHALL contain a unique identifier for the read point or business location, to the level of granularity that is intended to be communicated. This unique identifier SHALL be sufficient to distinguish one location from another. Extension elements added to `readPoint` or `bizLocation` SHALL NOT be required to distinguish one location from another.

Explanation (non-normative): This mechanism has been introduced as a short term measure to assist trading partners in exchanging master data about location identifiers. In the long term, it is expected that EPCIS events will include location identifiers, and information that describes the identifiers will be exchanged separately as master data. In the short term, however, the infrastructure to exchange location master data does not exist or is not widely implemented. In the absence of this infrastructure, extension elements within the events may be used to accompany each location identifier with its descriptive information. The standard SimpleEventQuery (Section 8.2.7.1) does not provide any direct means to use these extension elements to query for events. An industry group may determine that a given extension element is used to provide master data, in which case the master data features of the SimpleEventQuery (HASATTR and EQATTR) may be used in the query. It is up to an individual implementation to use the extension elements to populate whatever store is used to provide master data for the benefit of the query processor.

9.5 Schema for Core Event Types

The following is an XML Schema (XSD) for the Core Event Types data definition module. This schema imports additional schemas as shown in the following table:

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

In addition to the constraints implied by the schema, any value of type `xsd:dateTime` 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"`.

The XML Schema (XSD) for the Core Event Types data definition module is given below.:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:epcis="urn:epcglobal:epcis:xsd:1"
xmlns:sbdh="http://www.unece.org/cefact/namespaces/StandardBusinessDocumentHeader"
xmlns:epcglobal="urn:epcglobal:xsd:1" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="urn:epcglobal:epcis:xsd:1" elementFormDefault="unqualified"
attributeFormDefault="unqualified" version="1.1">
  <xsd:annotation>
```

```
    <xsd:documentation xml:lang="en">
      <epcglobal:copyright>Copyright (C) 2006-2013 GS1 AISBL, All Rights
Reserved.</epcglobal:copyright>
      <epcglobal:disclaimer>GS1 makes NO WARRANTY, EXPRESS OR IMPLIED, THAT THIS DOCUMENT IS
CORRECT, WILL NOT REQUIRE MODIFICATION AS EXPERIENCE AND TECHNOLOGY DICTATE, OR WILL BE SUITABLE
FOR ANY PURPOSE OR WORKABLE IN ANY APPLICATION, OR OTHERWISE.   Use of this document is with the
understanding that GS1 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO
ANY IMPLIED WARRANTY OF NON-INFRINGEMENT OF PATENTS OR COPYRIGHTS, MERCHANTABILITY AND/OR FITNESS
FOR A PARTICULAR PURPOSE, THAT THE INFORMATION IS ERROR FREE, NOR SHALL GS1 BE LIABLE FOR DAMAGES
OF ANY KIND, INCLUDING DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES,
ARISING OUT OF USE OR THE INABILITY TO USE INFORMATION CONTAINED HEREIN OR FROM ERRORS CONTAINED
HEREIN.</epcglobal:disclaimer>
      <epcglobal:specification>EPC INFORMATION SERVICE (EPCIS) Version
1.1</epcglobal:specification>
    </xsd:documentation>
  </xsd:annotation>
  <xsd:import namespace="urn:epcglobal:xsd:1" schemaLocation="./EPCglobal.xsd"/>
  <xsd:import namespace="http://www.unece.org/cefact/namespaces/StandardBusinessDocumentHeader"
schemaLocation="./StandardBusinessDocumentHeader.xsd"/>
  <!-- EPCIS CORE ELEMENTS -->
  <xsd:element name="EPCISDocument" type="epcis:EPCISDocumentType"/>
  <xsd:complexType name="EPCISDocumentType">
    <xsd:annotation>
      <xsd:documentation xml:lang="en">
      document that contains a Header and a Body.
              </xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
      <xsd:extension base="epcglobal:Document">
        <xsd:sequence>
          <xsd:element name="EPCISHeader" type="epcis:EPCISHeaderType" minOccurs="0"/>
          <xsd:element name="EPCISBody" type="epcis:EPCISBodyType"/>
          <xsd:element name="extension" type="epcis:EPCISDocumentExtensionType" minOccurs="0"/>
          <xsd:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:anyAttribute processContents="lax"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="EPCISDocumentExtensionType">
    <xsd:sequence>
      <xsd:any namespace="##local" processContents="lax" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:anyAttribute processContents="lax"/>
  </xsd:complexType>

  <xsd:complexType name="EPCISHeaderType">
    <xsd:annotation>
      <xsd:documentation xml:lang="en">
      specific header(s) including the Standard Business Document Header.
              </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element ref="sbdh:StandardBusinessDocumentHeader"/>
      <xsd:element name="extension" type="epcis:EPCISHeaderExtensionType" minOccurs="0"/>
      <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:anyAttribute processContents="lax"/>
  </xsd:complexType>
  <xsd:complexType name="EPCISHeaderExtensionType">
    <xsd:sequence>
      <xsd:any namespace="##local" processContents="lax" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:anyAttribute processContents="lax"/>
  </xsd:complexType>

  <xsd:complexType name="EPCISBodyType">
    <xsd:annotation>
      <xsd:documentation xml:lang="en">
      specific body that contains EPCIS related Events.
```

```
2415                </xsd:documentation>
2416          </xsd:annotation>
2417          <xsd:sequence>
2418            <xsd:element name="EventList" type="epcis:EventListType" minOccurs="0"/>
2419            <xsd:element name="extension" type="epcis:EPCISBodyExtensionType" minOccurs="0"/>
2420            <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2421          </xsd:sequence>
2422          <xsd:anyAttribute processContents="lax"/>
2423        </xsd:complexType>
2424        <xsd:complexType name="EPCISBodyExtensionType">
2425          <xsd:sequence>
2426            <xsd:any namespace="##local" processContents="lax" maxOccurs="unbounded"/>
2427          </xsd:sequence>
2428          <xsd:anyAttribute processContents="lax"/>
2429        </xsd:complexType>
2430
2431        <!-- EPCIS CORE ELEMENT TYPES -->
2432        <xsd:complexType name="EventListType">
2433          <xsd:choice minOccurs="0" maxOccurs="unbounded">
2434            <xsd:element name="ObjectEvent" type="epcis:ObjectEventType" minOccurs="0"
2435      maxOccurs="unbounded"/>
2436            <xsd:element name="AggregationEvent" type="epcis:AggregationEventType" minOccurs="0"
2437      maxOccurs="unbounded"/>
2438            <xsd:element name="QuantityEvent" type="epcis:QuantityEventType" minOccurs="0"
2439      maxOccurs="unbounded"/>
2440            <xsd:element name="TransactionEvent" type="epcis:TransactionEventType" minOccurs="0"
2441      maxOccurs="unbounded"/>
2442            <xsd:element name="extension" type="epcis:EPCISEventListExtensionType"/>
2443            <xsd:any namespace="##other" processContents="lax"/>
2444          </xsd:choice>
2445          <!-- Note: the use of "unbounded" in both the xsd:choice element
2446                     and the enclosed xsd:element elements is, strictly speaking,
2447                     redundant. However, this was found to avoid problems with
2448                     certain XML processing tools, and so is retained here.
2449                -->
2450        </xsd:complexType>
2451        <!-- Modified in 1.1 -->
2452        <xsd:complexType name="EPCISEventListExtensionType">
2453          <xsd:choice>
2454            <xsd:element name="TransformationEvent" type="epcis:TransformationEventType"/>
2455            <xsd:element name="extension" type="epcis:EPCISEventListExtension2Type"/>
2456          </xsd:choice>
2457        </xsd:complexType>
2458        <!-- Since 1.1 -->
2459        <xsd:complexType name="EPCISEventListExtension2Type">
2460          <xsd:sequence>
2461            <xsd:any namespace="##local" processContents="lax" maxOccurs="unbounded"/>
2462          </xsd:sequence>
2463          <xsd:anyAttribute processContents="lax"/>
2464        </xsd:complexType>
2465
2466        <xsd:complexType name="EPCListType">
2467          <xsd:sequence>
2468            <xsd:element name="epc" type="epcglobal:EPC" minOccurs="0" maxOccurs="unbounded"/>
2469          </xsd:sequence>
2470        </xsd:complexType>
2471        <xsd:simpleType name="ActionType">
2472          <xsd:restriction base="xsd:string">
2473            <xsd:enumeration value="ADD"/>
2474            <xsd:enumeration value="OBSERVE"/>
2475            <xsd:enumeration value="DELETE"/>
2476          </xsd:restriction>
2477        </xsd:simpleType>
2478        <xsd:simpleType name="ParentIDType">
2479          <xsd:restriction base="xsd:anyURI"/>
2480        </xsd:simpleType>
2481        <!-- Standard Vocabulary -->
2482        <xsd:simpleType name="BusinessStepIDType">
2483          <xsd:restriction base="xsd:anyURI"/>
2484        </xsd:simpleType>
2485        <!-- Standard Vocabulary -->
```

```
    <xsd:simpleType name="DispositionIDType">
      <xsd:restriction base="xsd:anyURI"/>
    </xsd:simpleType>
    <!-- User Vocabulary -->
    <xsd:simpleType name="EPCClassType">
      <xsd:restriction base="xsd:anyURI"/>
    </xsd:simpleType>
    <!-- Standard Vocabulary -->
    <!-- Since 1.1 -->
    <xsd:simpleType name="UOMType">
      <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
    <!-- Since 1.1 -->
    <xsd:complexType name="QuantityElementType">
      <xsd:sequence>
        <xsd:element name="epcClass" type="epcis:EPCClassType"/>
        <xsd:sequence minOccurs="0">
          <xsd:element name="quantity" type="xsd:decimal"/>
          <xsd:element name="uom" type="epcis:UOMType" minOccurs="0"/>
        </xsd:sequence>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="QuantityListType">
      <xsd:sequence>
        <xsd:element name="quantityElement" type="epcis:QuantityElementType" minOccurs="0"
maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>

    <!-- User Vocabulary -->
    <xsd:simpleType name="ReadPointIDType">
      <xsd:restriction base="xsd:anyURI"/>
    </xsd:simpleType>
    <xsd:complexType name="ReadPointType">
      <xsd:sequence>
        <xsd:element name="id" type="epcis:ReadPointIDType"/>
        <xsd:element name="extension" type="epcis:ReadPointExtensionType" minOccurs="0"/>
        <!-- The wildcard below provides the extension mechanism described in Section 9.4 -->
        <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="ReadPointExtensionType">
      <xsd:sequence>
        <xsd:any namespace="##local" processContents="lax" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:anyAttribute processContents="lax"/>
    </xsd:complexType>
    <!-- User Vocabulary -->
    <xsd:simpleType name="BusinessLocationIDType">
      <xsd:restriction base="xsd:anyURI"/>
    </xsd:simpleType>
    <xsd:complexType name="BusinessLocationType">
      <xsd:sequence>
        <xsd:element name="id" type="epcis:BusinessLocationIDType"/>
        <xsd:element name="extension" type="epcis:BusinessLocationExtensionType" minOccurs="0"/>
        <!-- The wildcard below provides the extension mechanism described in Section 9.4 -->
        <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="BusinessLocationExtensionType">
      <xsd:sequence>
        <xsd:any namespace="##local" processContents="lax" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:anyAttribute processContents="lax"/>
    </xsd:complexType>
    <!-- User Vocabulary -->
    <xsd:simpleType name="BusinessTransactionIDType">
      <xsd:restriction base="xsd:anyURI"/>
    </xsd:simpleType>
    <!-- Standard Vocabulary -->
    <xsd:simpleType name="BusinessTransactionTypeIDType">
```

```
      <xsd:restriction base="xsd:anyURI"/>
    </xsd:simpleType>
    <xsd:complexType name="BusinessTransactionType">
      <xsd:simpleContent>
        <xsd:extension base="epcis:BusinessTransactionIDType">
          <xsd:attribute name="type" type="epcis:BusinessTransactionTypeIDType" use="optional"/>
        </xsd:extension>
      </xsd:simpleContent>
    </xsd:complexType>
    <xsd:complexType name="BusinessTransactionListType">
      <xsd:sequence>
        <xsd:element name="bizTransaction" type="epcis:BusinessTransactionType"
maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
    <!-- User Vocabulary -->
    <!-- Since 1.1 -->
    <xsd:simpleType name="SourceDestIDType">
      <xsd:restriction base="xsd:anyURI"/>
    </xsd:simpleType>
    <!-- Standard Vocabulary -->
    <!-- Since 1.1 -->
    <xsd:simpleType name="SourceDestTypeIDType">
      <xsd:restriction base="xsd:anyURI"/>
    </xsd:simpleType>
    <!-- Since 1.1 -->
    <xsd:complexType name="SourceDestType">
      <xsd:simpleContent>
        <xsd:extension base="epcis:SourceDestIDType">
          <xsd:attribute name="type" type="epcis:SourceDestTypeIDType" use="required"/>
        </xsd:extension>
      </xsd:simpleContent>
    </xsd:complexType>
    <xsd:complexType name="SourceListType">
      <xsd:sequence>
        <xsd:element name="source" type="epcis:SourceDestType" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="DestinationListType">
      <xsd:sequence>
        <xsd:element name="destination" type="epcis:SourceDestType" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>

    <!-- User Vocabulary -->
    <!-- Since 1.1 -->
    <xsd:simpleType name="TransformationIDType">
      <xsd:restriction base="xsd:anyURI"/>
    </xsd:simpleType>

    <!-- Since 1.1 -->
    <xsd:complexType name="ILMDType">
      <xsd:sequence>
        <xsd:element name="extension" type="epcis:ILMDExtensionType" minOccurs="0"/>
        <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:anyAttribute processContents="lax"/>
    </xsd:complexType>
    <xsd:complexType name="ILMDExtensionType">
      <xsd:sequence>
        <xsd:any namespace="##local" processContents="lax" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:anyAttribute processContents="lax"/>
    </xsd:complexType>



    <!-- items listed alphabetically by name -->
    <!-- Some element types accommodate extensibility in the manner of
          "Versioning XML Vocabularies" by David Orchard (see
          http://www.xml.com/pub/a/2003/12/03/versioning.html).
```

```
2628
2629            In this approach, an optional <extension> element is defined
2630            for each extensible element type, where an <extension> element
2631            may contain future elements defined in the target namespace.
2632
2633            In addition to the optional <extension> element, extensible element
2634            types are declared with a final xsd:any wildcard to accommodate
2635            future elements defined by third parties (as denoted by the ##other
2636            namespace).
2637
2638            Finally, the xsd:anyAttribute facility is used to allow arbitrary
2639            attributes to be added to extensible element types. -->
2640      <xsd:complexType name="EPCISEventType" abstract="true">
2641        <xsd:annotation>
2642          <xsd:documentation xml:lang="en">
2643          base type for all EPCIS events.
2644          </xsd:documentation>
2645        </xsd:annotation>
2646        <xsd:sequence>
2647          <xsd:element name="eventTime" type="xsd:dateTime"/>
2648          <xsd:element name="recordTime" type="xsd:dateTime" minOccurs="0"/>
2649          <xsd:element name="eventTimeZoneOffset" type="xsd:string"/>
2650          <xsd:element name="baseExtension" type="epcis:EPCISEventExtensionType" minOccurs="0"/>
2651        </xsd:sequence>
2652        <xsd:anyAttribute processContents="lax"/>
2653      </xsd:complexType>
2654      <xsd:complexType name="EPCISEventExtensionType">
2655        <xsd:sequence>
2656          <xsd:any namespace="##local" processContents="lax" maxOccurs="unbounded"/>
2657        </xsd:sequence>
2658        <xsd:anyAttribute processContents="lax"/>
2659      </xsd:complexType>
2660
2661      <xsd:complexType name="ObjectEventType">
2662        <xsd:annotation>
2663          <xsd:documentation xml:lang="en">
2664          Object Event captures information about an event pertaining to one or more
2665          objects identified by EPCs.
2666                  </xsd:documentation>
2667        </xsd:annotation>
2668        <xsd:complexContent>
2669          <xsd:extension base="epcis:EPCISEventType">
2670            <xsd:sequence>
2671              <xsd:element name="epcList" type="epcis:EPCListType"/>
2672              <xsd:element name="action" type="epcis:ActionType"/>
2673              <xsd:element name="bizStep" type="epcis:BusinessStepIDType" minOccurs="0"/>
2674              <xsd:element name="disposition" type="epcis:DispositionIDType" minOccurs="0"/>
2675              <xsd:element name="readPoint" type="epcis:ReadPointType" minOccurs="0"/>
2676              <xsd:element name="bizLocation" type="epcis:BusinessLocationType" minOccurs="0"/>
2677              <xsd:element name="bizTransactionList" type="epcis:BusinessTransactionListType"
2678      minOccurs="0"/>
2679              <xsd:element name="extension" type="epcis:ObjectEventExtensionType" minOccurs="0"/>
2680              <xsd:any namespace="##other" processContents="lax" minOccurs="0"
2681      maxOccurs="unbounded"/>
2682            </xsd:sequence>
2683            <xsd:anyAttribute processContents="lax"/>
2684          </xsd:extension>
2685        </xsd:complexContent>
2686      </xsd:complexType>
2687      <!-- Modified in 1.1 -->
2688      <xsd:complexType name="ObjectEventExtensionType">
2689        <xsd:sequence>
2690          <xsd:element name="quantityList" type="epcis:QuantityListType" minOccurs="0"/>
2691          <xsd:element name="sourceList" type="epcis:SourceListType" minOccurs="0"/>
2692          <xsd:element name="destinationList" type="epcis:DestinationListType" minOccurs="0"/>
2693          <xsd:element name="ilmd" type="epcis:ILMDType" minOccurs="0"/>
2694          <xsd:element name="extension" type="epcis:ObjectEventExtension2Type" minOccurs="0"/>
2695        </xsd:sequence>
2696        <xsd:anyAttribute processContents="lax"/>
2697      </xsd:complexType>
2698      <!-- Since 1.1 -->
```

```
  <xsd:complexType name="ObjectEventExtension2Type">
    <xsd:sequence>
      <xsd:any namespace="##local" processContents="lax" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:anyAttribute processContents="lax"/>
  </xsd:complexType>

  <xsd:complexType name="AggregationEventType">
    <xsd:annotation>
      <xsd:documentation xml:lang="en">
      Aggregation Event captures an event that applies to objects that
      have a physical association with one another.
            </xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
      <xsd:extension base="epcis:EPCISEventType">
        <xsd:sequence>
          <xsd:element name="parentID" type="epcis:ParentIDType" minOccurs="0"/>
          <xsd:element name="childEPCs" type="epcis:EPCListType"/>
          <xsd:element name="action" type="epcis:ActionType"/>
          <xsd:element name="bizStep" type="epcis:BusinessStepIDType" minOccurs="0"/>
          <xsd:element name="disposition" type="epcis:DispositionIDType" minOccurs="0"/>
          <xsd:element name="readPoint" type="epcis:ReadPointType" minOccurs="0"/>
          <xsd:element name="bizLocation" type="epcis:BusinessLocationType" minOccurs="0"/>
          <xsd:element name="bizTransactionList" type="epcis:BusinessTransactionListType"
minOccurs="0"/>
          <xsd:element name="extension" type="epcis:AggregationEventExtensionType"
minOccurs="0"/>
          <xsd:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:anyAttribute processContents="lax"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <!-- Modified in 1.1 -->
  <xsd:complexType name="AggregationEventExtensionType">
    <xsd:sequence>
      <xsd:element name="childQuantityList" type="epcis:QuantityListType" minOccurs="0"/>
      <xsd:element name="sourceList" type="epcis:SourceListType" minOccurs="0"/>
      <xsd:element name="destinationList" type="epcis:DestinationListType" minOccurs="0"/>
      <xsd:element name="extension" type="epcis:AggregationEventExtension2Type" minOccurs="0"/>
    </xsd:sequence>
    <xsd:anyAttribute processContents="lax"/>
  </xsd:complexType>
  <!-- Since 1.1 -->
  <xsd:complexType name="AggregationEventExtension2Type">
    <xsd:sequence>
      <xsd:any namespace="##local" processContents="lax" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:anyAttribute processContents="lax"/>
  </xsd:complexType>

  <xsd:complexType name="QuantityEventType">
    <xsd:annotation>
      <xsd:documentation xml:lang="en">
      Quantity Event captures an event that takes place with respect to a specified quantity of
      object class.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
      <xsd:extension base="epcis:EPCISEventType">
        <xsd:sequence>
          <xsd:element name="epcClass" type="epcis:EPCClassType"/>
          <xsd:element name="quantity" type="xsd:int"/>
          <xsd:element name="bizStep" type="epcis:BusinessStepIDType" minOccurs="0"/>
          <xsd:element name="disposition" type="epcis:DispositionIDType" minOccurs="0"/>
          <xsd:element name="readPoint" type="epcis:ReadPointType" minOccurs="0"/>
          <xsd:element name="bizLocation" type="epcis:BusinessLocationType" minOccurs="0"/>
          <xsd:element name="bizTransactionList" type="epcis:BusinessTransactionListType"
minOccurs="0"/>
```

```
          <xsd:element name="extension" type="epcis:QuantityEventExtensionType" minOccurs="0"/>
          <xsd:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:anyAttribute processContents="lax"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="QuantityEventExtensionType">
    <xsd:sequence>
      <xsd:any namespace="##local" processContents="lax" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:anyAttribute processContents="lax"/>
  </xsd:complexType>

  <xsd:complexType name="TransactionEventType">
    <xsd:annotation>
      <xsd:documentation xml:lang="en">
      Transaction Event describes the association or disassociation of physical objects to one or
more business
      transactions.
            </xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
      <xsd:extension base="epcis:EPCISEventType">
        <xsd:sequence>
          <xsd:element name="bizTransactionList" type="epcis:BusinessTransactionListType"/>
          <xsd:element name="parentID" type="epcis:ParentIDType" minOccurs="0"/>
          <xsd:element name="epcList" type="epcis:EPCListType"/>
          <xsd:element name="action" type="epcis:ActionType"/>
          <xsd:element name="bizStep" type="epcis:BusinessStepIDType" minOccurs="0"/>
          <xsd:element name="disposition" type="epcis:DispositionIDType" minOccurs="0"/>
          <xsd:element name="readPoint" type="epcis:ReadPointType" minOccurs="0"/>
          <xsd:element name="bizLocation" type="epcis:BusinessLocationType" minOccurs="0"/>
          <xsd:element name="extension" type="epcis:TransactionEventExtensionType"
minOccurs="0"/>
          <xsd:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:anyAttribute processContents="lax"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <!-- Modified in 1.1 -->
  <xsd:complexType name="TransactionEventExtensionType">
    <xsd:sequence>
      <xsd:element name="quantityList" type="epcis:QuantityListType" minOccurs="0"/>
      <xsd:element name="sourceList" type="epcis:SourceListType" minOccurs="0"/>
      <xsd:element name="destinationList" type="epcis:DestinationListType" minOccurs="0"/>
      <xsd:element name="extension" type="epcis:TransactionEventExtension2Type" minOccurs="0"/>
    </xsd:sequence>
    <xsd:anyAttribute processContents="lax"/>
  </xsd:complexType>
  <!-- Since 1.1 -->
  <xsd:complexType name="TransactionEventExtension2Type">
    <xsd:sequence>
      <xsd:any namespace="##local" processContents="lax" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:anyAttribute processContents="lax"/>
  </xsd:complexType>

  <!-- Since 1.1 -->
  <xsd:complexType name="TransformationEventType">
    <xsd:annotation>
      <xsd:documentation xml:lang="en">
      Transformation Event captures an event in which inputs are consumed
      and outputs are produced
            </xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
      <xsd:extension base="epcis:EPCISEventType">
```

```
        <xsd:sequence>
          <xsd:element name="inputEPCList" type="epcis:EPCListType" minOccurs="0"/>
          <xsd:element name="inputQuantityList" type="epcis:QuantityListType" minOccurs="0"/>
          <xsd:element name="outputEPCList" type="epcis:EPCListType" minOccurs="0"/>
          <xsd:element name="outputQuantityList" type="epcis:QuantityListType" minOccurs="0"/>
          <xsd:element name="transformationID" type="epcis:TransformationIDType" minOccurs="0"/>
          <xsd:element name="bizStep" type="epcis:BusinessStepIDType" minOccurs="0"/>
          <xsd:element name="disposition" type="epcis:DispositionIDType" minOccurs="0"/>
          <xsd:element name="readPoint" type="epcis:ReadPointType" minOccurs="0"/>
          <xsd:element name="bizLocation" type="epcis:BusinessLocationType" minOccurs="0"/>
          <xsd:element name="bizTransactionList" type="epcis:BusinessTransactionListType"
minOccurs="0"/>
          <xsd:element name="sourceList" type="epcis:SourceListType" minOccurs="0"/>
          <xsd:element name="destinationList" type="epcis:DestinationListType" minOccurs="0"/>
          <xsd:element name="ilmd" type="epcis:ILMDType" minOccurs="0"/>
          <xsd:element name="extension" type="epcis:TransformationEventExtensionType"
minOccurs="0"/>
          <xsd:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:anyAttribute processContents="lax"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <!-- Since 1.1 -->
  <xsd:complexType name="TransformationEventExtensionType">
    <xsd:sequence>
      <xsd:any namespace="##local" processContents="lax" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:anyAttribute processContents="lax"/>
  </xsd:complexType>
</xsd:schema>
```

9.6 Core Event Types – Examples (non-normative)

This section provides examples of `EPCISDocuments`, rendered into XML [XML1.0].

9.6.1 Example 1 – Object Events with Instance-Level Identification

The example in this section contains two `ObjectEvents`, each containing instance-level identification. This example only uses features from EPCIS 1.0 and vocabulary from CBV 1.1. The second event shows an event-level vendor/user extension element named `myField`, following the method for vendor/user extensions specified in Section 9.1.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<epcis:EPCISDocument
    xmlns:epcis="urn:epcglobal:epcis:xsd:1"
    xmlns:example="http://ns.example.com/epcis"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    creationDate="2005-07-11T11:30:47.0Z"
    schemaVersion="1.1">
  <EPCISBody>
    <EventList>
      <ObjectEvent>
        <eventTime>2005-04-03T20:33:31.116-06:00</eventTime>
        <eventTimeZoneOffset>-06:00</eventTimeZoneOffset>
        <epcList>
          <epc>urn:epc:id:sgtin:0614141.107346.2017</epc>
          <epc>urn:epc:id:sgtin:0614141.107346.2018</epc>
        </epcList>
        <action>OBSERVE</action>
        <bizStep>urn:epcglobal:cbv:bizstep:shipping</bizStep>
        <disposition>urn:epcglobal:cbv:disp:in_transit</disposition>
        <readPoint>
          <id>urn:epc:id:sgln:0614141.07346.1234</id>
        </readPoint>
```

```
        <bizTransactionList>
            <bizTransaction
type="urn:epcglobal:cbv:btt:po">http://transaction.acme.com/po/12345678</bizTransaction>
        </bizTransactionList>
      </ObjectEvent>
      <ObjectEvent>
        <eventTime>2005-04-04T20:33:31.116-06:00</eventTime>
        <eventTimeZoneOffset>-06:00</eventTimeZoneOffset>
        <epcList>
          <epc>urn:epc:id:sgtin:0614141.107346.2018</epc>
        </epcList>
        <action>OBSERVE</action>
        <bizStep>urn:epcglobal:cbv:bizstep:receiving</bizStep>
        <disposition>urn:epcglobal:cbv:disp:in_progress</disposition>
        <readPoint>
           <id>urn:epc:id:sgln:0012345.11111.400</id>
        </readPoint>
        <bizLocation>
           <id>urn:epc:id:sgln:0012345.11111.0</id>
        </bizLocation>
        <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>
        <example:myField>Example of a vendor/user extension</example:myField>
      </ObjectEvent>
    </EventList>
  </EPCISBody>
</epcis:EPCISDocument>
```

9.6.2 Example 2 – Object Event with Class-Level Identification

The example in this section contains one `ObjectEvent`, containing only class-level identification. Note that the `<epcList>` element is still present, though empty, as this is required by the XML schema in order to maintain backward-compatibility with EPCIS 1.0. The `QuantityList`, along with other elements new in EPCIS 1.1, are all found in the `<extension>` area which is reserved for new features in EPCIS 1.1 (see Section 9.1). A vendor/user extension named `myField` is also included.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<epcis:EPCISDocument
    xmlns:epcis="urn:epcglobal:epcis:xsd:1"
    xmlns:example="http://ns.example.com/epcis"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    creationDate="2005-07-11T11:30:47.0Z"
    schemaVersion="1.1">
  <EPCISBody>
    <EventList>
      <ObjectEvent>
        <eventTime>2013-06-08T14:58:56.591Z</eventTime>
        <eventTimeZoneOffset>+02:00</eventTimeZoneOffset>
        <epcList/>
        <action>OBSERVE</action>
        <bizStep>urn:epcglobal:cbv:bizstep:receiving</bizStep>
        <disposition>urn:epcglobal:cbv:disp:in_progress</disposition>
        <readPoint>
          <id>urn:epc:id:sgln:0614141.00777.0</id>
        </readPoint>
        <bizLocation>
          <id>urn:epc:id:sgln:0614141.00888.0</id>
        </bizLocation>
        <extension>
          <quantityList>
            <quantityElement>
              <epcClass>urn:epc:class:lgtin:4012345.012345.998877</epcClass>
```

```
              <quantity>200</quantity>
              <uom>KGM</uom>
              <!-- Meaning: 200 kg of GTIN '04012345123456' belonging to lot '998877'-->
            </quantityElement>
          </quantityList>
          <sourceList>
            <source
type="urn:epcglobal:cbv:sdt:possessing_party">urn:epc:id:sgln:4012345.00001.0</source>
            <!-- Party which had physical possession at the originating endpoint of the business
transfer, e.g., a forwarder-->
            <source
type="urn:epcglobal:cbv:sdt:location">urn:epc:id:sgln:4012345.00225.0</source>
            <!-- Physical location of the originating endpoint, e.g., a distribution centre of
the forwarder-->
          </sourceList>
          <destinationList>
            <destination
type="urn:epcglobal:cbv:sdt:owning_party">urn:epc:id:sgln:0614141.00001.0</destination>
            <!-- Party which owns the physical objects at the terminating endpoint, e.g., a
retail company -->
            <destination
type="urn:epcglobal:cbv:sdt:location">urn:epc:id:sgln:0614141.00777.0</destination>
            <!-- Physical location of the terminating endpoint, e.g., a warehouse of the retail
company-->
          </destinationList>
        </extension>
        <example:myField>Example of a vendor/user extension</example:myField>
      </ObjectEvent>
    </EventList>
  </EPCISBody>
</epcis:EPCISDocument>
```

9.6.3 Example 3 – Aggregation Event with Mixed Identification

The example in this section contains one `AggregationEvent`, containing children having both instance-level and class-level identification. The `ChildQuantityList` is found in the `<extension>` area which is reserved for new features in EPCIS 1.1 (see Section 9.1). A vendor/user extension named `myField` is also included.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<epcis:EPCISDocument
    xmlns:epcis="urn:epcglobal:epcis:xsd:1"
    xmlns:example="http://ns.example.com/epcis"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    creationDate="2005-07-11T11:30:47.0Z"
    schemaVersion="1.1">
  <EPCISBody>
    <EventList>
      <AggregationEvent>
        <eventTime>2013-06-08T14:58:56.591Z</eventTime>
        <eventTimeZoneOffset>+02:00</eventTimeZoneOffset>
        <parentID>urn:epc:id:sscc:0614141.1234567890</parentID>
        <childEPCs>
          <epc>urn:epc:id:sgtin:0614141.107346.2017</epc>
          <epc>urn:epc:id:sgtin:0614141.107346.2018</epc>
        </childEPCs>
        <action>OBSERVE</action>
        <bizStep>urn:epcglobal:cbv:bizstep:receiving</bizStep>
        <disposition>urn:epcglobal:cbv:disp:in_progress</disposition>
        <readPoint>
          <id>urn:epc:id:sgln:0614141.00777.0</id>
        </readPoint>
        <bizLocation>
          <id>urn:epc:id:sgln:0614141.00888.0</id>
        </bizLocation>
        <extension>
          <childQuantityList>
            <quantityElement>
```

```
              <epcClass>urn:epc:idpat:sgtin:4012345.098765.*</epcClass>
              <quantity>10</quantity>
              <!-- Meaning: 10 units of GTIN '04012345987652' -->
            </quantityElement>
            <quantityElement>
              <epcClass>urn:epc:class:lgtin:4012345.012345.998877</epcClass>
              <quantity>200.5</quantity>
              <uom>KGM</uom>
              <!-- Meaning: 200.5 kg of GTIN '04012345123456' belonging to lot '998877'-->
            </quantityElement>
          </childQuantityList>
        </extension>
        <example:myField>Example of a vendor/user extension</example:myField>
      </AggregationEvent>
    </EventList>
  </EPCISBody>
</epcis:EPCISDocument>
```

9.6.4 Example 4 – Transformation Event

The example in this section contains one `TransformationEvent`, containing children having both instance-level and class-level identification. Instance/lot Master Data (ILMD) is also included, which describes the outputs of the transformation. A vendor/user extension named `myField` is also included. The entire event is wrapped in the `<extension>` element of EventList which is reserved for new event types in EPCIS 1.1 (see Section 9.1).

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<epcis:EPCISDocument schemaVersion="1.1" creationDate="2013-06-04T14:59:02.099+02:00"
xmlns:epcis="urn:epcglobal:epcis:xsd:1" xmlns:example="http://ns.example.com/epcis">
  <EPCISBody>
    <EventList>
     <extension>
      <TransformationEvent>
        <eventTime>2013-10-31T14:58:56.591Z</eventTime>
        <eventTimeZoneOffset>+02:00</eventTimeZoneOffset>
        <inputEPCList>
          <epc>urn:epc:id:sgtin:4012345.011122.25</epc>
          <epc>urn:epc:id:sgtin:4000001.065432.99886655</epc>
        </inputEPCList>
        <inputQuantityList>
          <quantityElement>
            <epcClass>urn:epc:class:lgtin:4012345.011111.4444</epcClass>
            <quantity>10</quantity>
            <uom>KGM</uom>
          </quantityElement>
          <quantityElement>
            <epcClass>urn:epc:class:lgtin:0614141.077777.987</epcClass>
            <quantity>30</quantity>
            <!-- As the uom field has been omitted, 30 instances (e.g., pieces) of GTIN
'00614141777778' belonging to lot '987' have been used. -->
          </quantityElement>
          <quantityElement>
            <epcClass>urn:epc:idpat:sgtin:4012345.066666.*</epcClass>
            <quantity>220</quantity>
            <!-- As the uom field has been omitted and as an EPC pattern is indicated, 220
instances (e.g., pieces) of GTIN '04012345666663' have been used. -->
          </quantityElement>
        </inputQuantityList>
        <outputEPCList>
          <epc>urn:epc:id:sgtin:4012345.077889.25</epc>
          <epc>urn:epc:id:sgtin:4012345.077889.26</epc>
          <epc>urn:epc:id:sgtin:4012345.077889.27</epc>
          <epc>urn:epc:id:sgtin:4012345.077889.28</epc>
        </outputEPCList>
        <bizStep>urn:epcglobal:cbv:bizstep:transforming</bizStep>
        <disposition>urn:epcglobal:cbv:disp:in_progress</disposition>
        <readPoint>
```

```
          <id>urn:epc:id:sgln:4012345.00001.0</id>
        </readPoint>
        <ilmd>
          <!-- Section, in which the instance/ lot master data referring to the objects indicated
in the outputEPCList are defined.-->
          <example:bestBeforeDate>2014-12-10</example:bestBeforeDate>
          <!-- The namespace 'example' is just a placeholder for the domain under which the ILMD
attributes are defined (for instance, by a GS1 working group). Meaning: the best before date of
the above GTIN + lot is the 10th December 2014. -->
          <example:batch>XYZ</example:batch>
        </ilmd>
        <example:myField>Example of a vendor/user extension</example:myField>
      </TransformationEvent>
    </extension>
  </EventList>
 </EPCISBody>
</epcis:EPCISDocument>
```

9.7 Schema for Master Data

The following is an XML Schema (XSD) defining the XML binding of master data for the Core Event Types data definition module. This schema is only used for returning results from the `SimpleMasterDataQuery` query type (Section 8.2.7.2). This schema imports additional schemas as shown in the following table:

Namespace	Location Reference	Source
`urn:epcglobal:xsd:1`	EPCglobal.xsd	Section 9.3
`http://www.unece.org/cefact/namespaces/StandardBusinessDocumentHeader`	StandardBusinessDocumentHeader.xsd	UN/CEFACT web site; see Section 9.2
`urn:epcglobal:epcis:xsd:1`	EPCglobal-epcis-1_1.xsd	Section 9.5

In addition to the constraints implied by the schema, any value of type `xsd:dateTime` 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 XML Schema (XSD) for master data is given below.:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:epcismd="urn:epcglobal:epcis-masterdata:xsd:1"
   xmlns:sbdh="http://www.unece.org/cefact/namespaces/StandardBusinessDocumentHeader"
   xmlns:epcglobal="urn:epcglobal:xsd:1"
   xmlns:epcis="urn:epcglobal:epcis:xsd:1"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   targetNamespace="urn:epcglobal:epcis-masterdata:xsd:1"
   elementFormDefault="unqualified"
   attributeFormDefault="unqualified"
   version="1.0">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">
      <epcglobal:copyright>Copyright (C) 2006, 2005, 2004 EPCglobal Inc., All Rights
Reserved.</epcglobal:copyright>
```

```
      <epcglobal:disclaimer>EPCglobal Inc., its members, officers, directors, employees, or
agents shall not be liable for any injury, loss, damages, financial or otherwise, arising from,
related to, or caused by the use of this document.  The use of said document shall constitute
your express consent to the foregoing exculpation.</epcglobal:disclaimer>
      <epcglobal:specification>EPC INFORMATION SERVICE (EPCIS) Version
1.0</epcglobal:specification>
    </xsd:documentation>
  </xsd:annotation>
  <xsd:import namespace="urn:epcglobal:xsd:1" schemaLocation="./EPCglobal.xsd"/>
  <xsd:import
     namespace="http://www.unece.org/cefact/namespaces/StandardBusinessDocumentHeader"
     schemaLocation="./StandardBusinessDocumentHeader.xsd"/>
  <xsd:import
    namespace="urn:epcglobal:epcis:xsd:1"
    schemaLocation="./EPCglobal-epcis-1_1.xsd"/>

  <!-- MasterData CORE ELEMENTS -->
  <xsd:element name="EPCISMasterDataDocument" type="epcismd:EPCISMasterDataDocumentType"/>
  <xsd:complexType name="EPCISMasterDataDocumentType">
    <xsd:annotation>
      <xsd:documentation xml:lang="en">
      MasterData document that contains a Header and a Body.
              </xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
      <xsd:extension base="epcglobal:Document">
        <xsd:sequence>
          <xsd:element name="EPCISHeader" type="epcis:EPCISHeaderType" minOccurs="0"/>
          <xsd:element name="EPCISBody" type="epcismd:EPCISMasterDataBodyType"/>
          <xsd:element name="extension" type="epcismd:EPCISMasterDataDocumentExtensionType"
minOccurs="0"/>
          <xsd:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:anyAttribute processContents="lax"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="EPCISMasterDataBodyType">
    <xsd:annotation>
      <xsd:documentation xml:lang="en">
      MasterData specific body that contains Vocabularies.
              </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element name="VocabularyList" type="epcismd:VocabularyListType" minOccurs="0"/>
      <xsd:element name="extension" type="epcismd:EPCISMasterDataBodyExtensionType"
minOccurs="0"/>
      <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:anyAttribute processContents="lax"/>
  </xsd:complexType>

  <!-- MasterData CORE ELEMENT TYPES -->
  <xsd:complexType name="VocabularyListType">
    <xsd:sequence>
      <xsd:element name="Vocabulary" type="epcismd:VocabularyType" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="VocabularyType">
    <xsd:sequence>
      <xsd:element name="VocabularyElementList" type="epcismd:VocabularyElementListType"
minOccurs="0"/>
      <xsd:element name="extension" type="epcismd:VocabularyExtensionType" minOccurs="0"/>
      <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="type" type="xsd:anyURI" use="required"/>
    <xsd:anyAttribute processContents="lax"/>
```

```
    </xsd:complexType>

    <xsd:complexType name="VocabularyElementListType">
      <xsd:sequence>
        <xsd:element name="VocabularyElement" type="epcismd:VocabularyElementType"
maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>

    <!-- Implementations SHALL treat a <children list containing zero elements
         in the same way as if the <children> element were omitted altogether.
    -->
    <xsd:complexType name="VocabularyElementType">
      <xsd:sequence>
        <xsd:element name="attribute" type="epcismd:AttributeType" minOccurs="0"
maxOccurs="unbounded"/>
        <xsd:element name="children" type="epcismd:IDListType" minOccurs="0"/>
        <xsd:element name="extension" type="epcismd:VocabularyElementExtensionType" minOccurs="0"/>
        <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="id" type="xsd:anyURI" use="required"/>
      <xsd:anyAttribute processContents="lax"/>
    </xsd:complexType>

    <xsd:complexType name="AttributeType">
      <xsd:complexContent>
        <xsd:extension base="xsd:anyType">
          <xsd:attribute name="id" type="xsd:anyURI" use="required"/>
          <xsd:anyAttribute processContents="lax"/>
        </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>

    <xsd:complexType name="IDListType">
      <xsd:sequence>
        <xsd:element name="id" type="xsd:anyURI" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:anyAttribute processContents="lax"/>
    </xsd:complexType>

    <xsd:complexType name="EPCISMasterDataDocumentExtensionType">
      <xsd:sequence>
        <xsd:any namespace="##local" processContents="lax" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:anyAttribute processContents="lax"/>
    </xsd:complexType>

    <xsd:complexType name="EPCISMasterDataHeaderExtensionType">
      <xsd:sequence>
        <xsd:any namespace="##local" processContents="lax" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:anyAttribute processContents="lax"/>
    </xsd:complexType>

    <xsd:complexType name="EPCISMasterDataBodyExtensionType">
      <xsd:sequence>
        <xsd:any namespace="##local" processContents="lax" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:anyAttribute processContents="lax"/>
    </xsd:complexType>

    <xsd:complexType name="VocabularyExtensionType">
      <xsd:sequence>
        <xsd:any namespace="##local" processContents="lax" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:anyAttribute processContents="lax"/>
    </xsd:complexType>

    <xsd:complexType name="VocabularyElementExtensionType">
      <xsd:sequence>
        <xsd:any namespace="##local" processContents="lax" maxOccurs="unbounded"/>
```

```
    </xsd:sequence>
    <xsd:anyAttribute processContents="lax"/>
  </xsd:complexType>
</xsd:schema>
```

9.8 Master Data – Example (non-normative)

Here is an example `EPCISMasterDataDocument` containing master data for `BusinessLocation` and `ReadPoint` vocabularies, rendered into XML [XML1.0]:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<epcismd:EPCISMasterDataDocument
   xmlns:epcismd="urn:epcglobal:epcis-masterdata:xsd:1"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   schemaVersion="1.0"
   creationDate="2005-07-11T11:30:47.0Z">
  <EPCISBody>
    <VocabularyList>
      <Vocabulary type="urn:epcglobal:epcis:vtype:BusinessLocation">
        <VocabularyElementList>
          <VocabularyElement id="urn:epc:id:sgln:0037000.00729.0">
            <attribute id="http://epcis.example.com/mda/latitude">+18.0000</attribute>
            <attribute id="http://epcis.example.com/mda/longitude">-70.0000</attribute>
            <attribute id="http://epcis.example.com/mda/address">
              <example:Address xmlns:example="http://epcis.example.com/ns">
                <Street>100 Nowhere Street</Street>
                <City>Fancy</City>
                <State>DC</State>
                <Zip>99999</Zip>
              </example:Address>
            </attribute>
            <children>
              <id>urn:epc:id:sgln:0037000.00729.8201</id>
              <id>urn:epc:id:sgln:0037000.00729.8202</id>
              <id>urn:epc:id:sgln:0037000.00729.8203</id>
            </children>
          </VocabularyElement>
          <VocabularyElement id="urn:epc:id:sgln:0037000.00729.8201">
            <attribute id="urn:epcglobal:cbv:mda:sst">201</attribute>
          </VocabularyElement>
          <VocabularyElement id="urn:epc:id:sgln:0037000.00729.8202">
            <attribute id="urn:epcglobal:cbv:mda:sst">202</attribute>
            <children>
              <id>urn:epc:id:sgln:0037000.00729.8203</id>
            </children>
          </VocabularyElement>
          <VocabularyElement id="urn:epc:id:sgln:0037000.00729.8203">
            <attribute id="urn:epcglobal:cbv:mda:sst">202</attribute>
            <attribute id="urn:epcglobal:cbv:mda:ssa">402</attribute>
          </VocabularyElement>
        </VocabularyElementList>
      </Vocabulary>
      <Vocabulary type="urn:epcglobal:epcis:vtype:ReadPoint">
        <VocabularyElementList>
          <VocabularyElement id="urn:epc:id:sgln:0037000.00729.8201">
            <attribute id="urn:epcglobal:cbv:mda:site">0037000007296</attribute>
            <attribute id="urn:epcglobal:cbv:mda:sst">201</attribute>
          </VocabularyElement>
          <VocabularyElement id="urn:epc:id:sgln:0037000.00729.8202">
            <attribute id="urn:epcglobal:cbv:mda:site">0037000007296</attribute>
            <attribute id="urn:epcglobal:cbv:mda:sst">202</attribute>
          </VocabularyElement>
          <VocabularyElement id="urn:epc:id:sgln:0037000.00729.8203">
            <attribute id="urn:epcglobal:cbv:mda:site">0037000007296</attribute>
            <attribute id="urn:epcglobal:cbv:mda:sst">203</attribute>
          </VocabularyElement>
        </VocabularyElementList>
      </Vocabulary>
    </VocabularyList>
```

```
  </EPCISBody>
</epcismd:EPCISMasterDataDocument>
```

10 Bindings for Core Capture Operations Module

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

10.1 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 Section 8.1.2 by delivering a message to the endpoint provided by the capture server. The message SHALL be one of the following:

- an XML document whose root element conforms to the `EPCISDocument` element as defined by the schema of Section 9.5; or
- an XML document whose root element conforms to the `EPCISQueryDocument` element as defined by the schema of Section 11.1, 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 authorization failure as described in Section 8.1.1, or for some other reason. The specific circumstances under which a positive or negative acknowledgement are indicated is implementation-dependent. All implementations, however, SHALL either accept all events in the message or reject all events.

10.2 HTTP Binding

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

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

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

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

- an XML document whose root element conforms to the `EPCISDocument` element as defined by the schema of Section 9.5; or
- an XML document whose root element conforms to the `EPCISQueryDocument` element as defined by the schema of Section 11.1, where the element immediately nested within the `EPCISBody` element is a `QueryResults` element, and where the `resultsBody` element within the `QueryResults` element contains an `EventList` element.

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

Status codes returned by the capture server SHALL conform to [RFC2616], Section 10. In particular, the capture server SHALL return status code 200 to indicate successful completion of the capture operation, and any status code 3xx, 4xx, or 5xx SHALL indicate that the capture operation was not successfully completed.

11 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)	Section 11.2
	XML over AS2	Section 11.3
Query Callback Interface	XML over HTTP	Section 11.4.2
	XML over HTTP+TLS (HTTPS)	Section 11.4.3
	XML over AS2	Section 11.4.4

All of these bindings share a common XML syntax, specified in Section 11.1. 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 Section 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 Section 8.2.6
- XML elements for the Query Callback Interface as defined in Section 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 Section 9.5. Each binding specifies its own rules for using this header, if applicable.

11.1 XML Schema for Core Query Operations Module

The following schema 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 Section 11. 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 below is used to represent a query parameter as used by the `poll` and `subscribe` methods of the query interface defined in Section 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
Int	`xsd:integer`
Float	`xsd:double`
Time	`xsd:dateTime`
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 Section 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	Section 9.3

Namespace	Location Reference	Source
`http://www.unece.org/cefact/namespaces/StandardBusinessDocumentHeader`	StandardBusinessDocumentHeader.xsd	UN/CEFACT web site; see Section 9.2
`urn:epcglobal:epcis:xsd:1`	EPCglobal-epcis-1_1.xsd	Section 9.5
`urn:epcglobal:epcis-masterdata:xsd:1`	EPCglobal-epcis-masterdata-1_1.xsd	Section 9.7

In addition to the constraints implied by the schema, any value of type `xsd:dateTime` 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 XML Schema (XSD) for the Core Query Operations Module is given below.:

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema targetNamespace="urn:epcglobal:epcis-query:xsd:1"
   xmlns:epcis="urn:epcglobal:epcis:xsd:1"
   xmlns:epcismd="urn:epcglobal:epcis-masterdata:xsd:1"
   xmlns:epcisq="urn:epcglobal:epcis-query:xsd:1"
   xmlns:epcglobal="urn:epcglobal:xsd:1"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="unqualified"
   attributeFormDefault="unqualified"
   version="1.0">

  <xsd:annotation>
    <xsd:documentation xml:lang="en">
      <epcglobal:copyright>
         Copyright (C) 2006, 2005 EPCglobal Inc., All Rights Reserved.
      </epcglobal:copyright>
      <epcglobal:disclaimer>
            EPCglobal Inc, its members, officers, directors, employees, or
            agents shall not be liable for any injury, loss, damages, financial
            or otherwise, arising from, related to, or caused by the use of
            this document.  The use of said document shall constitute your
            express consent to the foregoing exculpation.
      </epcglobal:disclaimer>
      <epcglobal:specification>
         EPCIS Query 1.0
      </epcglobal:specification>
    </xsd:documentation>
  </xsd:annotation>

  <xsd:import namespace="urn:epcglobal:xsd:1" schemaLocation="./EPCglobal.xsd"/>
  <xsd:import namespace="urn:epcglobal:epcis:xsd:1" schemaLocation="./EPCglobal-epcis-1_1.xsd"/>
  <xsd:import namespace="urn:epcglobal:epcis-masterdata:xsd:1" schemaLocation="./EPCglobal-epcis-
masterdata-1_1.xsd"/>

  <xsd:element name="EPCISQueryDocument" type="epcisq:EPCISQueryDocumentType"/>
  <xsd:complexType name="EPCISQueryDocumentType">
     <xsd:complexContent>
        <xsd:extension base="epcglobal:Document">
          <xsd:sequence>
            <xsd:element name="EPCISHeader" type="epcis:EPCISHeaderType" minOccurs="0"/>
            <xsd:element name="EPCISBody" type="epcisq:EPCISQueryBodyType"/>
```