EPC Information Services (EPCIS) Version 1.0 Specification

Ratified Standard

April 12, 2007

Disclaimer

EPCglobal Inc™ is providing this document as a service to interested industries. This document was developed through a consensus process of interested parties. Although efforts have been to assure that the document is correct, reliable, and technically accurate, EPCglobal Inc makes NO WARRANTY, EXPRESS OR IMPLIED, THAT THIS DOCUMENT IS CORRECT, WILL NOT REQUIRE MODIFICATION AS EXPERIENCE AND TECHNOLOGICAL ADVANCES DICTATE, OR WILL BE SUITABLE FOR ANY PURPOSE OR WORKABLE IN ANY APPLICATION, OR OTHERWISE.

Use of this document is with the understanding that EPCglobal Inc has no liability for any claim to the contrary, or for any damage or loss of any kind or nature.

Abstract

This document is an EPCglobal normative specification that defines Version 1.0 of EPC Information Services (EPCIS). The goal of EPCIS is to enable disparate applications to leverage Electronic Product Code (EPC) data via EPC-related data sharing, both within and across enterprises. Ultimately, this sharing is aimed at enabling participants in the EPCglobal Network to gain a shared view of the disposition of EPC-bearing 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 EPCglobal. See `www.epcglobalinc.org` for more information. This version of this document has been ratified by the EPCglobal Board as of April 12, 2007.

Comments on this document should be sent to the EPCglobal Software Action Group mailing list sag_epcis2_wg@lists.epcglobalinc.org.

Table of Contents

1 Introduction

This document is an EPCglobal normative specification that defines Version 1.0 of EPC Information Services (EPCIS). The goal of EPCIS is to enable disparate applications to leverage Electronic Product Code (EPC) data via EPC-related data sharing, both within and across enterprises. Ultimately, this sharing is aimed at enabling participants in the EPCglobal Network to gain a shared view of the disposition of EPC-bearing objects within a relevant business context.

This Version 1.0 specification is intended to provide a basic capability that meets the above goal. In particular, this specification is designed to meet the requirements of a basic set of use cases that the user community has identified as a minimal useful set. Other use cases and capabilities are expected to be addressed through follow-on versions of this specification, and companion specifications.

The scope of this Version 1.0 specification has been guided by an informative document produced by a prior EPCglobal working group, titled "EPC Information Services (EPCIS) User Definition" [EPCIS-User]. Several of the relevant sections are quoted below. Readers should refer to this document for a discussion of the use cases that have guided the design decisions embodied in this specification.

1.1 Services Approach

(This section is mostly quoted from [EPCIS-User].)

The objective of EPCIS as stated above is obviously very broad, implying that the "S" in EPCIS stands for **EPC Information Sharing**. The intent of this broad objective is to encompass the widest possible set of use cases and to not overly constrain the technical approaches for addressing them.

That said, our experience since starting to define EPCIS indicates that attempting to be so broad is confusing and distracting, especially with regard to the technical approaches. For example, this objective could be partially addressed by making existing B2B transactions such as Advanced Shipment Notices (ASNs) and Receipt Advices "EPC enabled." It could also be addressed by defining a new "Services-based" approach to enable EPC-related data sharing. And there are no doubt other possible alternatives. Because these alternatives call for different development approaches and likely involve different groups of people, it has been difficult to define a path forward.

To get past this confusion, this specification focuses on an **EPC Information Service** approach, recognizing that some of what must be defined in this approach (such as data element standards) will be applicable to other approaches as well. The **EPC Information Service** approach will define a **standard interface** to enable EPC-related data to be **captured** and **queried** using a defined set of **service operations** and associated EPC-related **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 EPC-related 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 specifes only a standard data sharing interface between applications that capture EPC-related 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 EPCISs 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 EPC Information Service.

2 Relationship to the EPCglobal Architecture Framework

(This section is largely quoted from [EPCIS-User] and [EPCAF])

As depicted in the diagram below, EPCIS sits at the highest level of the EPCglobal Architecture Framework, both above the level of raw EPC observations (e.g., the Tag Protocol and the Reader "Wireline" Protocol), as well as above the level of filtered, consolidated observations (e.g., the Filtering & Collection Interface). In the diagram, the plain green bars denote interfaces governed by EPCglobal standards, while the blue shadowed boxes denote roles played by hardware and/or software components of the system.

(A single physical software or hardware component may play more than one role. For example, a "smart reader" may perform middleware functions and expose the ALE interface as its external interface. In that case, the "reader" (the metal box with the antenna) is playing both the Reader and Middleware role in the diagram, and the Reader Protocol Interface is internal to the smart reader (if it exists at all). Likewise, it is common to have enterprise applications such as Warehouse Management Systems that simultaneously play the role of EPCIS Capturing Application (e.g. detecting EPCs during product movement during truck loading), an EPCIS-enabled Repository (e.g. recording case-to-pallet associations), and an EPCIS Accessing Application (e.g. carrying out business decisions based on EPCIS-level data).)

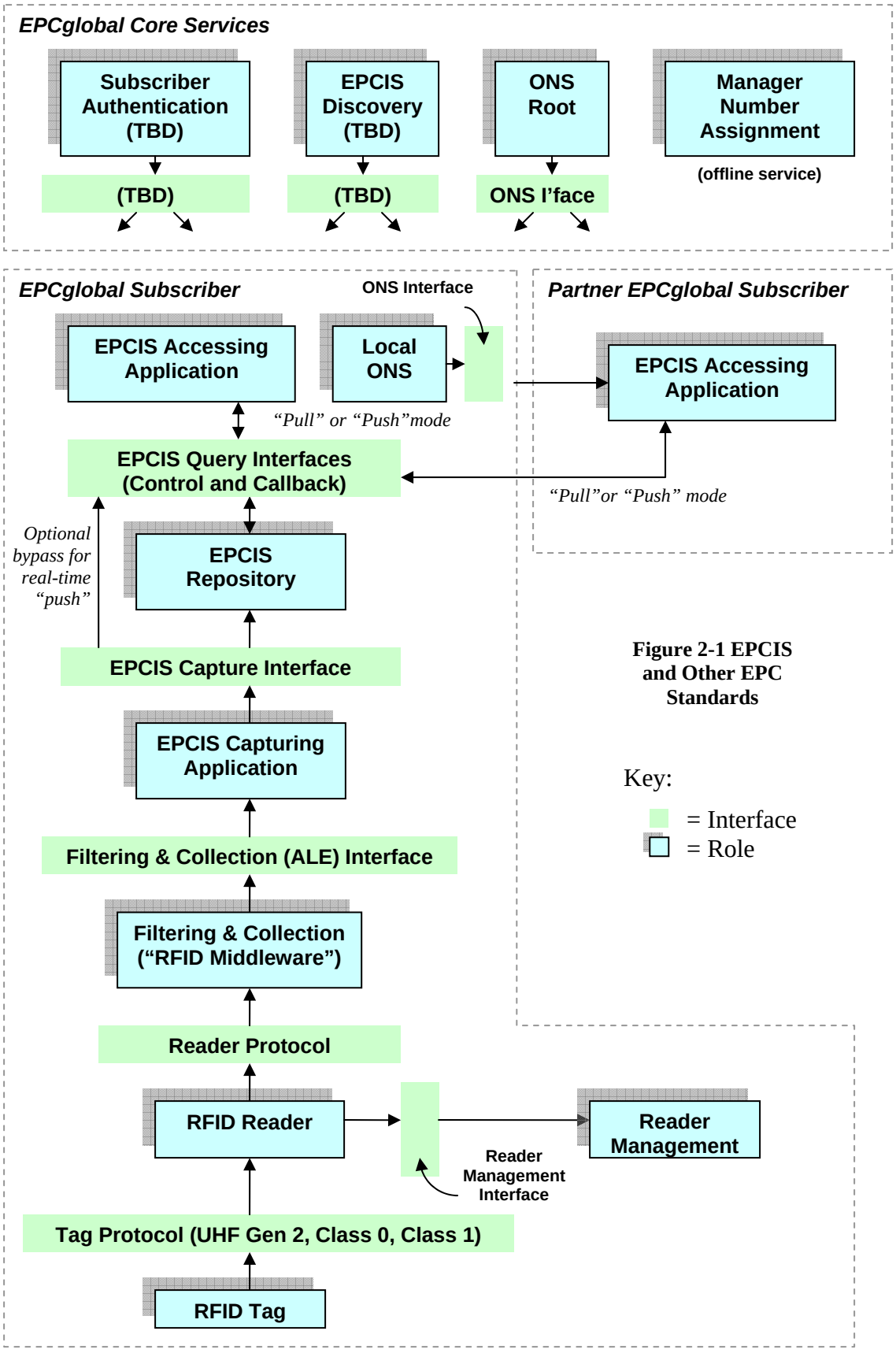


Figure 2-1 EPCIS and Other EPC Standards

189

While EPCIS is an integral part of the EPCglobal Network, it differs from elements at the lower layers of the Architecture in three key respects:

1. EPCIS deals explicitly with historical data (in addition to current data). The lower layers of the stack, in contrast, are oriented exclusively towards real-time processing of EPC data.

2. EPCIS often deals not just with raw EPC observations, but also in contexts that imbue those observations with meaning relative to the physical world and to specific steps in operational or analytical business processes. The lower layers of the stack are more purely observational in nature. An EPCIS-level event, while containing much of the same EPC data as a Filtering & Collection event, is at a semantically higher level because it incorporates an understanding of the business context in which the EPC data were obtained. Moreover, there is no requirement that an EPCIS event be directly related to a specific physical tag observation. For example, an EPCIS Quantity Event (Section 7.2.11) contains information that may be generated purely by software, such as an inventory application.

3. EPCIS operates within enterprise IT environments at a level that is much more diverse and multi-faceted than the lower levels of the EPCglobal Network Architecture. 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 EPCglobal Network 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).

More specifically, the following outlines the responsibilities of each element of the EPCglobal Architecture Framework. Further information may be found in [EPCAF], from which the diagram above and the following text is quoted.

- *Readers* Make multiple observations of RFID tags while they are in the read zone.
- *Reader Protocol Interface* Defines the control and delivery of raw tag reads from 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 tag reads, over time intervals delimited by events defined by the EPCIS Capturing Application (e.g. tripping a motion detector).
- *Filtering & Collection (ALE) Interface* Defines the control and delivery of filtered and collected tag read data from the Filtering & Collection role to the EPCIS Capturing Application 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.
- *EPCIS Capturing Application* 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 EPCIS Capturing Application may, for example, coordinate a conveyor system with Filtering & Collection events, may check for exceptional conditions and take corrective action (e.g., diverting a bad case into a rework area), may present information to a human operator, and so on. The EPCIS Capturing Application understands the business process step or steps during which EPCIS data capture takes place. This role may be complex, involving the association of multiple Filtering & Collection events 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 "smart shelves" deployed that generate periodic observations about objects that enter or leave the shelf. Here, the Filtering & Collection-level event and the EPCIS-level event may be so similar that no actual processing at the EPCIS Capturing Application level is necessary, and the EPCIS Capturing Application merely configures and routes events from the Filtering & Collection interface directly to an EPCIS-enabled Repository.

- *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 "optional 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.

- *ONS* ONS is a network service that is used to look up pointers to EPCIS Repositories, starting from an EPC Manager Number or full Electronic Product Code. Specifically, ONS provides a means to look up a pointer to the EPCIS service provided by the organization who commissioned the EPC of the object in question. The most common example is where ONS is used to discover an EPCIS service that contains product data from a manufacturer for a given EPC. ONS may also be used to discover an EPCIS service that has master data pertaining to a particular EPCIS location identifier (this use case is not yet fully addressed in the ONS specification).

- *Discovery Capability* Refers to a mechanism, not yet defined at the time of this writing, for locating all EPCIS-enabled Repositories that might have data about a particular EPC. This is useful when the relevant EPCIS services might not otherwise be known to the party who wishes to query them, such as when the handling history of an object is desired but not known (e.g. in support of track-and-trace across a multi-party supply chain). The initial work to define EPCglobal's approach towards adding Discovery Capability to the EPCglobal Architecture Framework is currently underway within the EPCglobal Architecture Review Committee.

The interfaces within this stack are designed to insulate the higher levels of the stack 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 Reader Protocol Interface insulates the higher layers from knowing what RF protocols are in use, and what reader makes/models have been chosen. If a different reader is substituted, the information at the Reader Protocol Interface remains the same.

- The Filtering & Collection Interface insulates the higher layers from the physical design choices made regarding how tags are sensed and accumulated, and how the time boundaries of events are triggered. If a single four-antenna 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 likely corresponds 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 EPC codes. 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.

In summary, EPCIS-level data differs from lower layers in the EPCglobal Network Architecture by incorporating semantic information about the business process in which EPC 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 at the lower layers of the EPCglobal Network Architecture. The historical nature implies that EPCIS interfaces will need a richer set of access techniques than the ALE or Reader Protocol interfaces. The incorporation of operational or business process context into EPCIS implies that EPCIS will traffic in a richer set of data types, and moreover will need 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 specifications must be layered carefully so that even when EPCIS interfaces with 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 specifications at different layers.

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.

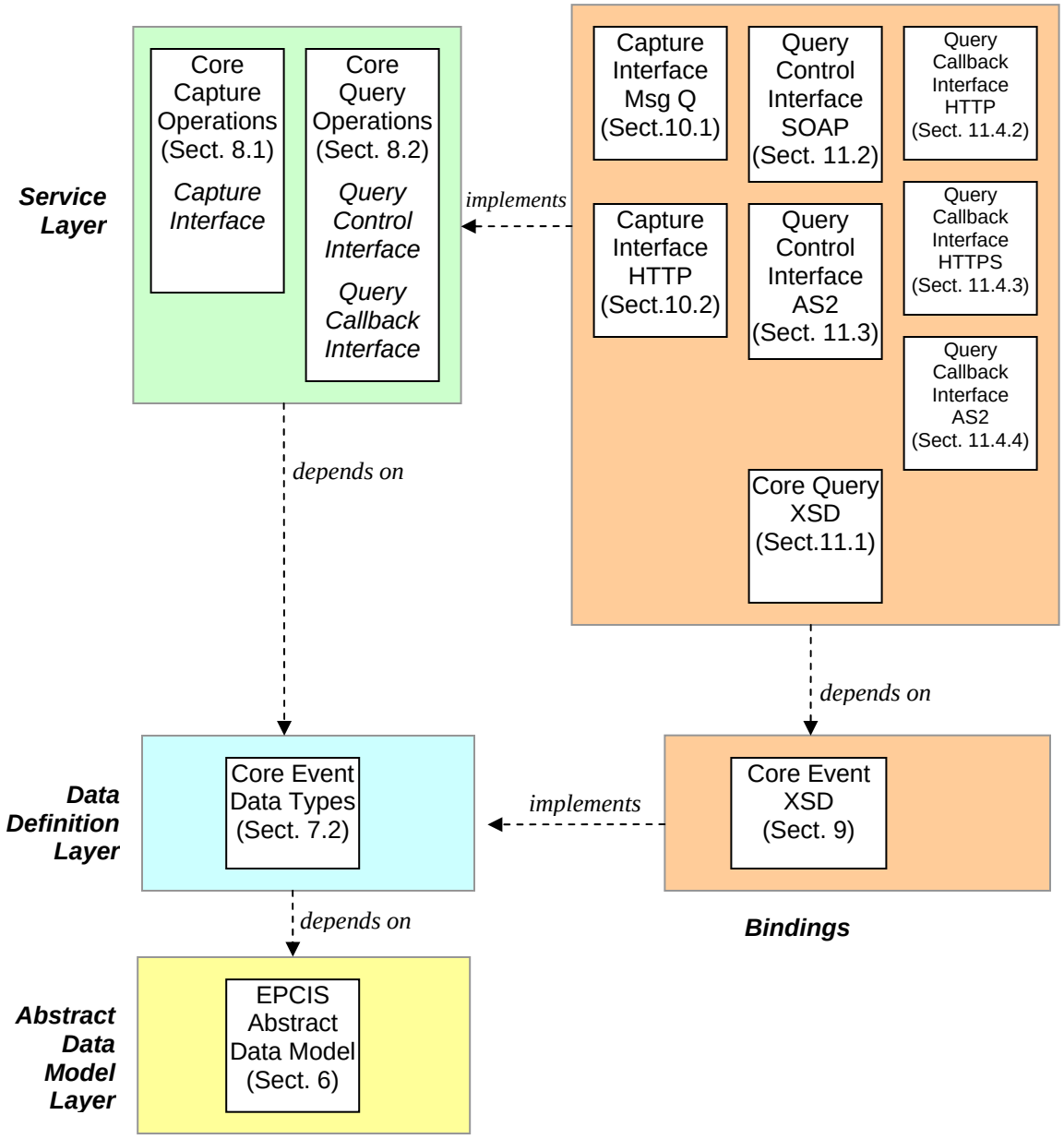
➢ Placeholders for changes that need to be made to this document prior to its reaching the final stage of approved EPCglobal specification are prefixed by a rightward-facing arrowhead, as this paragraph is.

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.

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 created 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 catalog service that traffics in data defined in the relevant data definition modules. This product may conform to as many as six different EPCIS specifications: the data definition module that describes product catalog 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 catalog 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

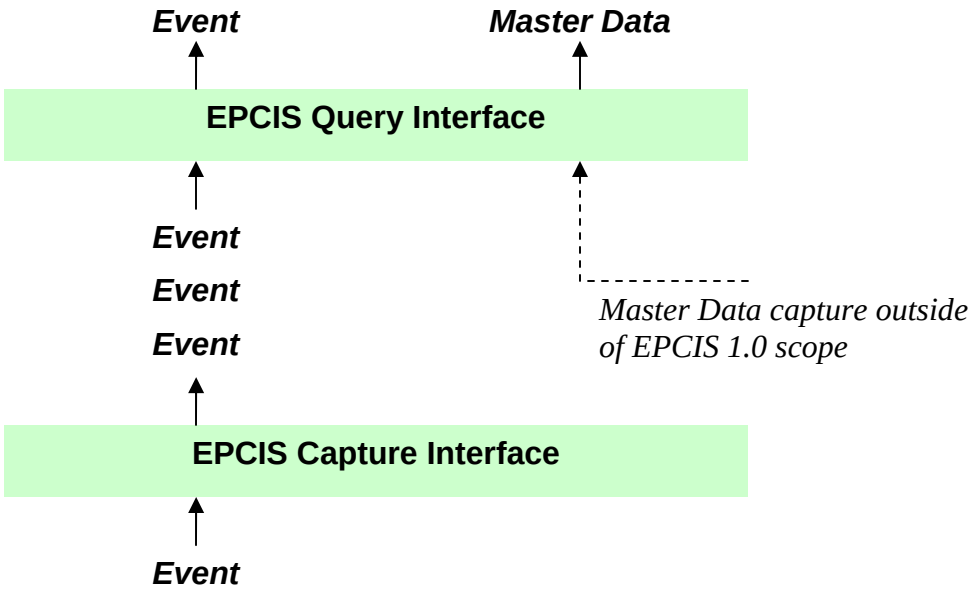
This section gives a normative description of the abstract data model that underlies EPCIS.

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 1.0 specification.

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

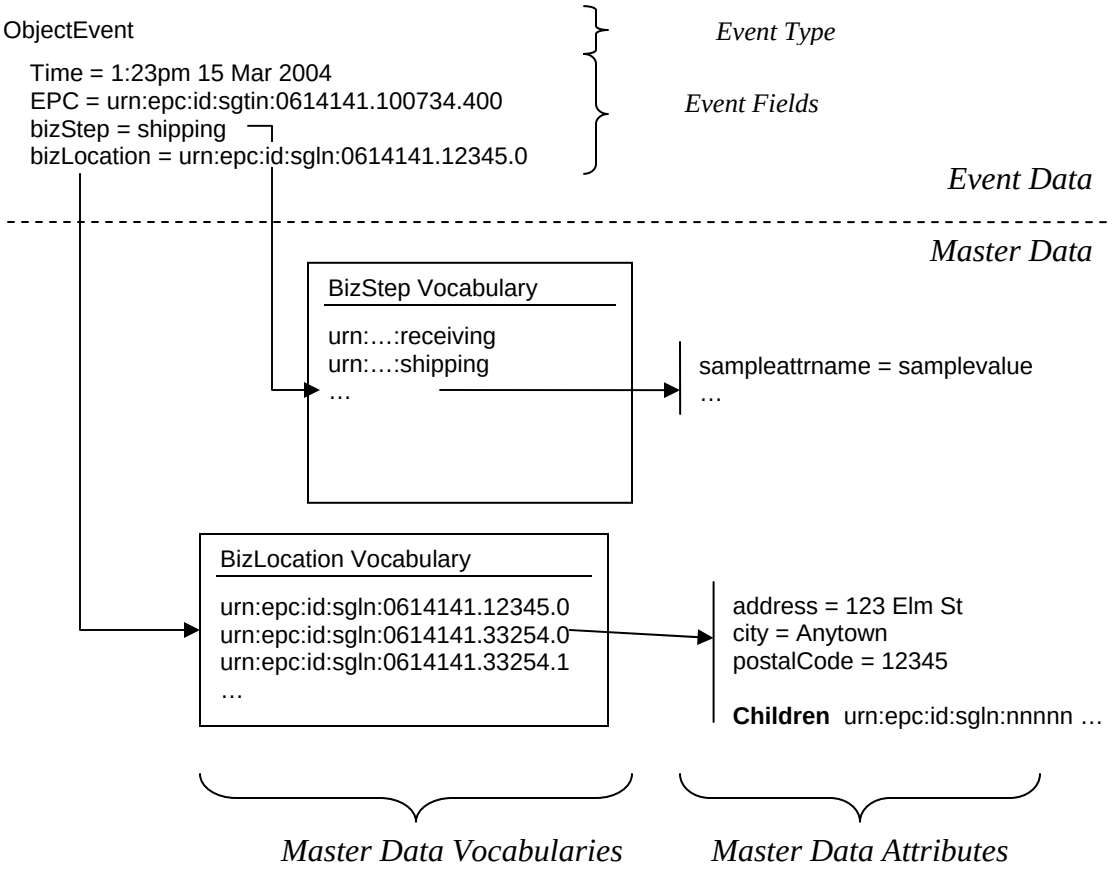
These relationships are illustrated below:



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

Explanation (non-normative): While for the purposes of this specification the terms "event data" and "master data" mean nothing more than "data that fits the structure provided here," the structures defined in the Abstract Data Model Layer are designed to provide an appropriate representation for data commonly requiring exchange through EPCIS within industries seeking to exploit the EPCglobal Network. Informally, these two types of data may be understood as follows. Event data grows in quantity as more business is transacted, and refers to things that happen at specific moments in time. An example of event data is "At 1:23pm on 15 March 2004, EPC X was observed at Location L." Master data does not generally grow merely because more business is transacted (though master data does tend to grow as organizations grow in size), is not typically tied to specific moments in time (though master data may change slowly over time), and provides interpretation for elements of event data. An example of master data is "Location L refers to the distribution center located at 123 Elm Street, Anytown, US." All of the data in the set of use cases considered in the creation of the EPCIS 1.0 specification can be modeled as a combination of event data and master data of this kind.

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



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

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

used to model sets such as the set of available location names, the set of available business process step names, and so on.

- *Vocabulary Element* An identifier that names one of the alternatives modeled 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 within EPCIS to model conceptual and physical 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 EPCglobal, industry consortia outside EPCglobal, 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 Center #3." Acme Corp may generate an event having a business location of "Acme Corp. Distribution Center #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.

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 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 kinds of organizations who will wish to extend the EPCIS specification, as summarized below:

Organization Type	**Extension Method**					**How Disseminated**
	New Event Type	**New Event Field**	**New Vocab Type**	**New Master Data Attr**	**New Vocab Element**	
EPCglobal EPCIS Working Group	Yes	Yes	Yes	Occasionally	Rarely	New Version of EPCIS Spec
EPCglobal Business Action Group for a specific industry	Rarely	Rarely	Occasionally	Yes	Yes (Standard Vocabulary)	Specification Document

Organization Type	Extension Method					How Disseminated
	New Event Type	New Event Field	New Vocab Type	New Master Data Attr	New Vocab Element	
Industry Consortium or Private End User Group outside EPCglobal	Rarely	Rarely	Occasionally	Yes	Yes (Standard Vocabulary)	Private Group Interoperability Specification
Individual End User	Rarely	Rarely	Rarely	Rarely	Yes (User vocabulary)	Updated Master Data via EPCIS Query or other data sync

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 [TDS1.3, Section 4.1]. 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 EPC codes [TDS1.3]. The owning authority for these URNs is EPCglobal.

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 Center #3" would have a children list including the element that means "Acme Corp. Distribution Center #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:

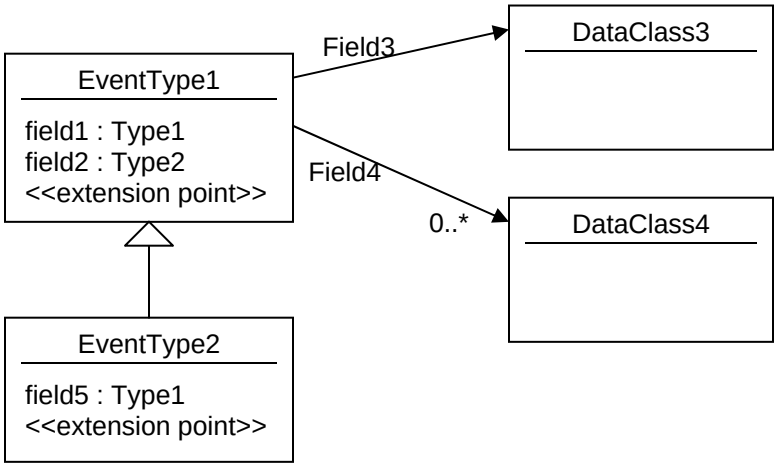
- *Value Types* Definitions of data types that are used to describe the values of Event Fields and of Master Data Attributes. The Core Event Types Module defines the primitive types that are available for use by all Data Definition Modules. Each Vocabulary that is defined is also implicitly a Value Type.
- *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 EPCglobal 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 supercede 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 supercede the event in question.

The retrospective/prospective dichotomy plays an important role in EPCIS’s definition of location, in Section 7.2.3.

7.2 Core Event Types Module

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

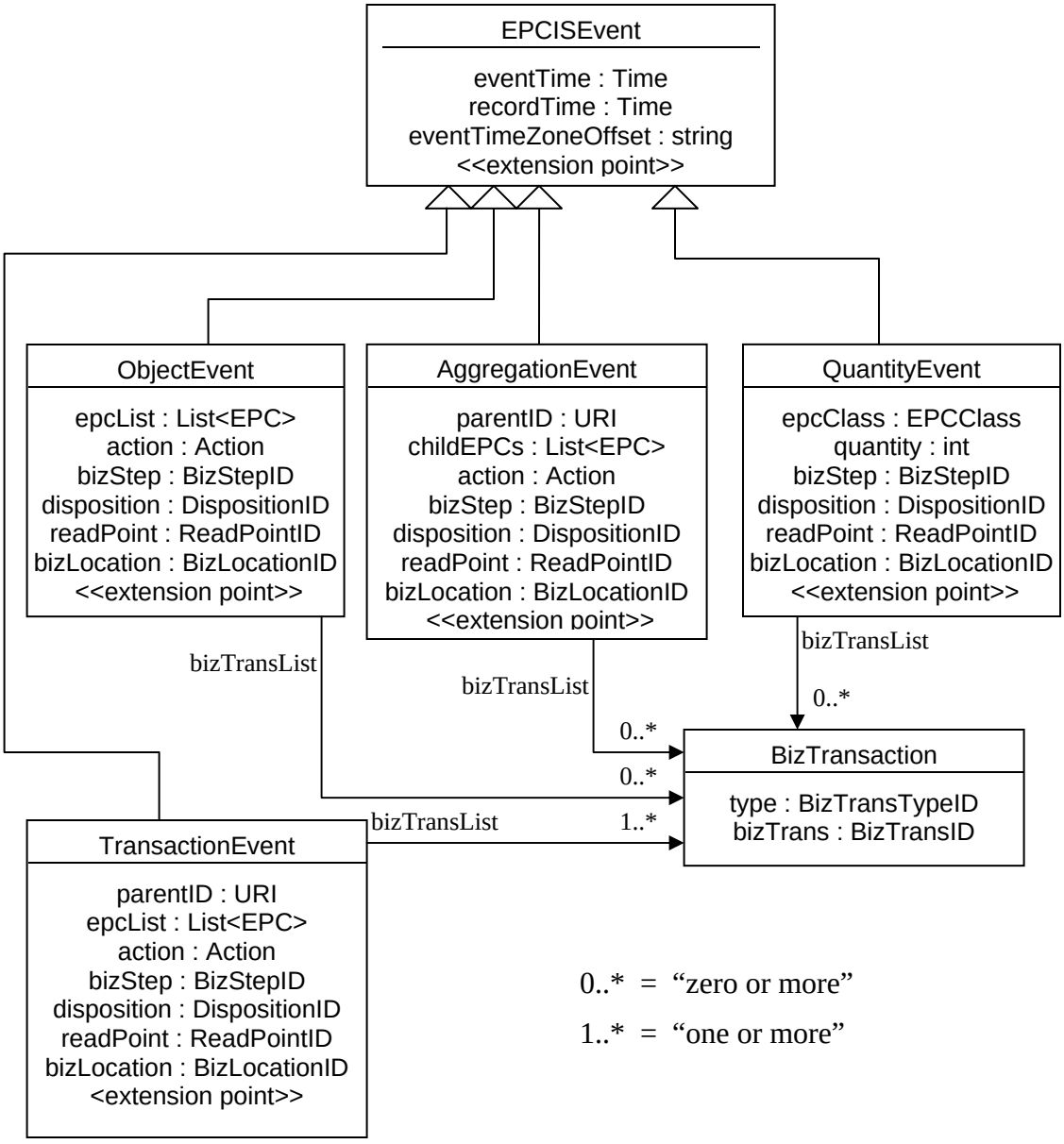
The components of this module, following the outline given in Section 7.1.1, are as follows:

- *Value Types* Primitive types defined in Section 7.2.1.
- *Event Types* Event types as shown in the UML diagram below, and defined in Sections 7.2.8 through 7.2.12.
- *Event Fields* Included as part of the Event Types definitions.
- *Vocabulary Types* Types defined in Sections 7.2.3 through 7.2.7, 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 7.2.4), the `Disposition` vocabulary (Section 7.2.5), and the `BusinessTransactionType` vocabulary (Section 7.2.6.1).

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

- `EPCISEvent` (Section 7.2.8) is a generic base class for all event types in this module as well as others.
- `ObjectEvent` (Section 7.2.9) represents an event that happened to one or more entities denoted by EPCs.
- `AggregationEvent` (Section 7.2.10) represents an event that happened to one or more entities denoted by EPCs 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 7.2.11) represents an event concerned with a specific quantity of entities sharing a common EPC class, but where the individual identities of the entities are not specified.
- `TransactionEvent` (Section 7.2.12) represents an event in which one or more entities denoted by EPCs become associated or disassociated with one or more identified business transactions.

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

814

815 Each of the core event types (not counting the generic `EPCISEvent`) has fields that
816 represent four key dimensions of any EPCIS event. These four dimensions are: (1) the
817 object(s) or other entities that are the subject of the event; (2) the date and time; (3) the
818 location at which the event occurred; (4) the business context. These four dimensions
819 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 a `QuantityEvent` the "what" dimension is an `EPCClass` and a count). 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 (`QuantityEvent`) `BusinessTransactionList` (`TransactionEvent`)	
When	`Time`	
Where	`ReadPointID`	`BusinessLocationID`
Why (business context)	`BusinessStepID`	`DispositionID`

In addition to the fields belonging to the four key dimensions, events may carry additional descriptive information in other fields. In this specification, the only descriptive field is the `bizTransactionList` field of `ObjectEvent` and `AggregationEvent`, which in each case indicates that the event occurred within the context of a particular business transaction. (The `bizTransactionList` field of `TransactionEvent`, however, is not "additional descriptive information," but rather the primary subject (the "what" dimension) of the event.) 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`	7.2.3	User	`urn:epcglobal:epcis:vtype:ReadPoint`
`BusinessLocationID`	7.2.3	User	`urn:epcglobal:epcis:vtype:BusinessLocation`
`BusinessStepID`	7.2.4	Standard	`urn:epcglobal:epcis:vtype:BusinessStep`
`DispositionID`	7.2.5	Standard	`urn:epcglobal:epcis:vtype:Disposition`

Vocabulary Type	Section	User / Standard	URI
BusinessTransaction	7.2.6.2	User	urn:epcglobal:epcis:vtype:BusinessTransaction
BusinessTrasactionTypeID	7.2.6.1	Standard	urn:epcglobal:epcis:vtype:BusinessTransactionType
EPCClass	7.2.7	User	urn:epcglobal:epcis:vtype:EPCClass

840

841 7.2.1 Primitive Types

842 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 [TDS1.3]. Unless otherwise noted, EPCs are represented in "pure identity" URI form as defined in [TDS1.3], Section 4.1.

843

844 The EPC type is defined as a primitive type for use in events when referring to EPCs that
845 are not part of a Vocabulary Type. For example, an SGTIN EPC used to denote an
846 instance of a trade item in the `epcList` field of an `ObjectEvent` is an instance of the
847 EPC primitive type. But an SGLN EPC used as a read point identifier (Section 7.2.3) in
848 the `ReadPoint` field of an `ObjectEvent` is a Vocabulary Element, not an instance of
849 the EPC primitive type.

850 *Explanation (non-normative): This reflects a design decision not to consider individual*
851 *trade item instances as Vocabulary Elements having Master Data, owing to the fact that*
852 *trade item instances are constantly being created and hence new EPCs representing*
853 *trade items are constantly being commissioned. In part, this design decision reflects*
854 *consistent treatment of Master Data as excluding data that grows as more business is*
855 *transacted (see comment in Section 6.1), and in part reflects the pragmatic reality that*
856 *data about trade item instances is likely to be managed more like event data than master*
857 *data when it comes to aging, database design, etc.*

858 7.2.2 Action Type

859 The `Action` type says how an event relates to the lifecycle of the entity being described.
860 For example, `AggregationEvent` (Section 7.2.10) is used to capture events related to
861 physical 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.

7.2.3 Location Types

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 traveling;

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.)

The reader/location types are as follows:

	Type	Description
Primitive Reader Types – *not* location types for EPCIS		
	`PhysicalReaderID`	This is the serialized 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

	Type	Description
		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 "how or where the EPCIS event was detected."
	`BusinessLocationID`	A Business Location is a uniquely identified and discretely recorded location that is meant to designate the specific place where an object is assumed to be following an EPCIS event until it is reported to be at a different Business Location by a subsequent EPCIS event. As with the Read Point, the EPCIS Capturing Application determines the Business Location based on whatever means it chooses. Conceptually, the Business Location is designed to identify "where the object is 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 [TDS1.3] SHALL be used.

Illustration (non-normative): For example, in industries governed by EAN.UCC General Specifications, `readPointID`, and `businessLocationID` may be SGLN-URIs [TDS1.3, Section 4.3.5], and `physicalReaderID` may be an SGTIN-URI [TDS1.3, Section 4.3.3].

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

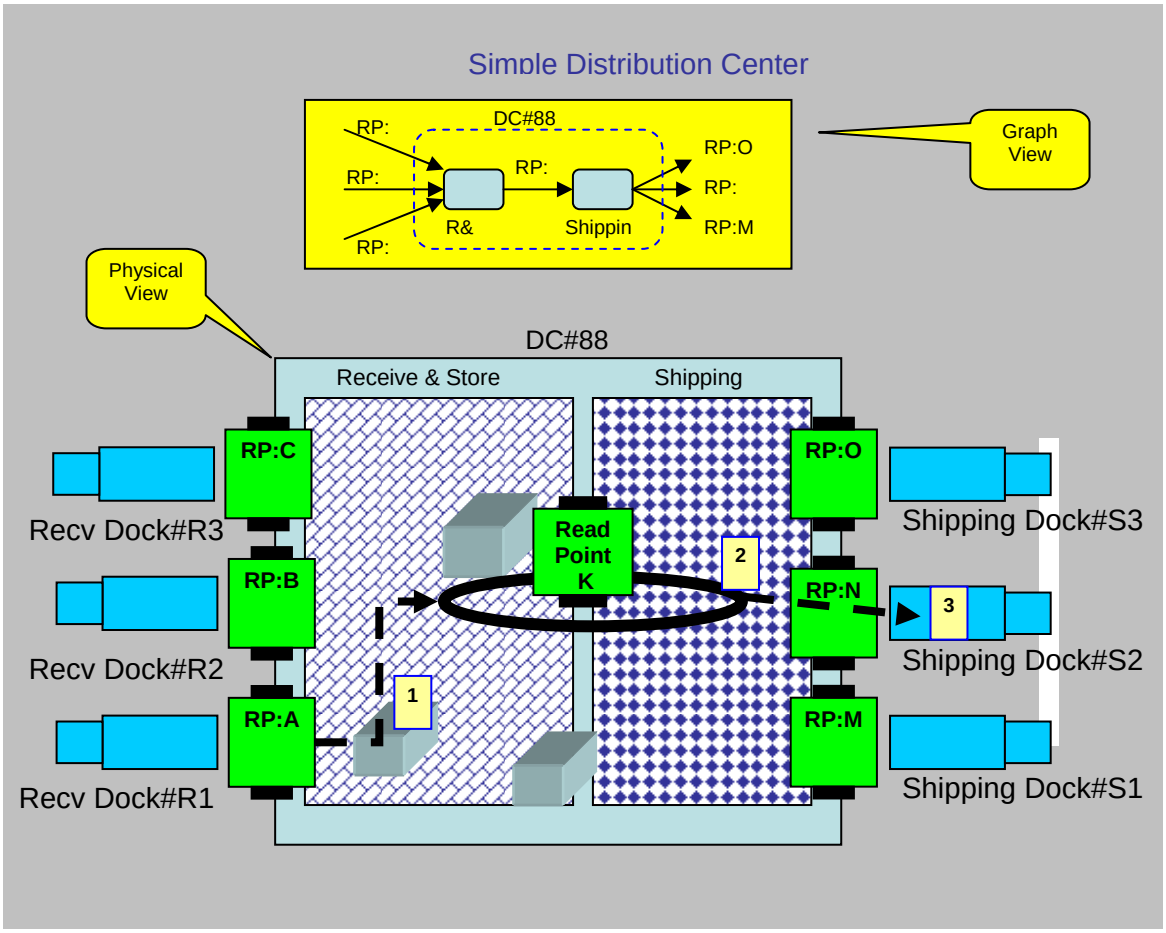
Explanation (non-normative): Allowing non-EPC URIs for locations gives organizations greater freedom to reuse existing ways of naming locations.

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.

7.2.3.1 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"*	*DC#88.Transit*	*Product loaded on truck 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 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.

7.2.4 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 EPCglobal standards 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.

7.2.5 Disposition

`DispositionID` is a vocabulary whose elements denote a business state of an object. An example is an identifier that denotes "available for sale." 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 EPCglobal standards to define some common terms such as "available for sale" or "in storage," 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.

7.2.6 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.

7.2.6.1 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 EPCglobal 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.

7.2.6.2 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.

7.2.7 EPCClass

`EPCClass` is a Vocabulary whose elements denote classes of trade items. `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.

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

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

where `CompanyPrefix` is an EAN.UCC Company Prefix (including leading zeros) and `ItemRefAndIndicator` consists of the indicator digit of a GTIN followed by the digits of the item reference of a 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 `ItemRefAndIndicator` fields, and having any serial number whatsoever.

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

7.2.8 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.
`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` 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, followed by a colon character '`:`', followed by two digits. 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 7.2.1 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.*

7.2.9 ObjectEvent (subclass of EPCISEvent)

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

While more than one EPC may appear in an `ObjectEvent`, no relationship or association between those EPCs 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 EPC(s) named in the event. Specifically:

`Action` value	**Meaning**
`ADD`	The EPC(s) named in the event have been commissioned as part of this event; that is, the EPC(s) have been issued and associated with an object (s) for the first time.
`OBSERVE`	The event represents a simple observation of the EPC(s) named in the event, not their commissioning or decommissioning.
`DELETE`	The EPC(s) named in the event have been decommissioned as part of this event; that is, the EPC(s) do not exist subsequent to the event and should not be observed again.

1107 Fields:

Field	Type	Description
`eventTime` `recordTime`	(Inherited from `EPCISEvent`; see Section 7.2.8)	
`epcList`	`List<EPC>`	An unordered list of one or more EPCs naming the physical objects to which the event pertained. Each element of this list SHALL be a URI [RFC2396] denoting the unique identity for a physical 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 [TDS1.3], Section 4.1. Implementations MAY accept URI-formatted identifiers other than EPCs.
`action`	`Action`	How this event relates to the lifecycle of the EPCs named in this event. See above for more detail.
`bizStep`	`BusinessStepID`	(Optional) The business step of which this event was a part.
`disposition`	`DispositionID`	(Optional) The business condition of the objects associated with the EPCs, presumed to hold 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 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.

Retrospective semantics:

- An event described by `bizStep` (and any other fields) took place with respect to each EPC in `epcList` at `eventTime` at location `readPoint`.
- (If `action` is `ADD`) The EPCs in `epcList` were commissioned (issued for the first time).
- (If `action` is `DELETE`) The EPCs in `epcList` were decommissioned (retired from future use).
- (If `action` is `ADD` and a non-empty `bizTransactionList` is specified) An association exists between the business transactions enumerated in `bizTransactionList` and the EPCs in `epcList`.
- (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`.

Prospective semantics:

- (If `action` is `ADD`) The EPCs in `epcList` may appear in subsequent events.
- (If `action` is `DELETE`) The EPCs in `epcList` should not appear in subsequent events.
- (If `disposition` is specified) The business condition of the objects associated with the EPCs in `epcList` is as described by `disposition`.
- (If `disposition` is omitted) The business condition of the objects associated with the EPCs in `epcList` is unchanged.

- (If `bizLocation` is specified) The physical objects associated with the EPCs in `epcList` are at business location `bizLocation`.
- (If `bizLocation` is omitted) The business location of the physical objects associated with the EPCs in `epcList` is unknown.
- (If `action` is `ADD` and a non-empty `bizTransactionList` is specified) An association exists between the business transactions enumerated in `bizTransactionList` and the EPCs in `epcList`.

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 a ObjectEvent with a non-empty `bizTransactionList` does not cause a TransactionEvent to be returned from a query.

7.2.10 AggregationEvent (subclass of EPCISEvent)

The event type `AggregationEvent` describes events that apply to objects that have been physically 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 physical 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 EPCs named 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.

`Action` value	Meaning
`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` 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 EPCs named 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. The list of child EPCs may be omitted from the `AggregationEvent`, which means that *all* children have been disaggregated. (This permits dissaggregation when the event capture software does not know the identities of all the children.)

1162

1163 The `AggregationEvent` type includes fields that refer to a single "parent" (often a
1164 "containing" entity) and one or more "children" (often "contained" objects). A parent
1165 identifier is required when `action` is `ADD` or `DELETE`, but optional when `action` is
1166 `OBSERVE`.

1167 *Explanation (non-normative): A parent identifier is used when `action` is `ADD` so that*
1168 *there is a way of referring to the association in subsequent events when `action` is*
1169 *`DELETE`. The parent identifier is optional when `action` is `OBSERVE` because the*
1170 *parent is not always known during an intermediate observation. For example, a pallet*
1171 *receiving process may rely on RFID tags to determine the EPCs of cases on the pallet,*
1172 *but there might not be an RFID tag for the pallet (or if there is one, it may be*
1173 *unreadable).*

1174 The `AggregationEvent` is intended to indicate aggregations among physical objects,
1175 and so the children are identified by EPCs. The parent entity, however, is not necessarily
1176 a physical object that's separate from the aggregation itself, and so the parent is identified
1177 by an arbitrary URI, which MAY be an EPC, but MAY be another identifier drawn from
1178 a suitable private vocabulary.

1179 *Explanation (non-normative): In many manufacturing operations, for example, it is*
1180 *common to create a load several steps before an EPC for the load is assigned. In such*
1181 *situations, an internal tracking number (often referred to as a "license plate number," or*
1182 *LPN) is assigned at the time the load is created, and this is used up to the point of*
1183 *shipment. At the point of shipment, an SSCC code (which* is *an EPC) is assigned. In*
1184 *EPCIS, this would be modeled by (a) an `AggregateEvent` with `action` equal to*
1185 *`ADD` at the time the load is created, and (b) a second `AggregationEvent` with*
1186 *`action` equal to `ADD` at the time the SSCC is assigned (the first association may also be*
1187 *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`	(Inherited from `EPCISEvent`; see Section 7.2.8)	
`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 [TDS1.3], Section 4.1.
`childEPCs`	`List<EPC>`	An unordered list of the EPCs of the contained objects. Each element of the list SHALL be a URI [RFC2396] denoting the unique identity for a physical object. When the unique identity is an Electronic Product Code, the list element SHALL be the "pure identity" URI for the contained EPC as specified in [TDS1.3], Section 4.1. Implementations MAY accept URI-formatted identifiers other than EPCs. The `childEPCs` list MAY be empty if `action` is `DELETE`, indicating that all children are disaggregated from the parent.

Field	Type	Description
`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 until contradicted by a subsequent event.
`readPoint`	`ReadPointID`	(Optional) The read point at which the event took place.
`bizLocation`	`BusinessLocationID`	(Optional) The business location where the objects associated with the containing and contained EPCs may be found, until contradicted by a subsequent event.
`bizTransactionList`	Unordered list of zero or more `BusinessTransaction` instances	(Optional) An unordered list of business transactions that define the context of this event.

Retrospective semantics:

- An event described by `bizStep` (and any other fields) took place involving containing entity `parentID` and the contained EPCs in `childEPCs`, at `eventTime` and location `readPoint`.
- (If `action` is `ADD`) The EPCs in `childEPCs` were aggregated to containing entity `parentID`.
- (If `action` is `DELETE` and `childEPCs` is non-empty) The EPCs in `childEPCs` were disaggregated from `parentID`.
- (If `action` is `DELETE` and `childEPCs` is empty) All contained EPCs 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 EPCs in `childEPCs`, 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`.

Prospective semantics:

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

7.2.11 QuantityEvent (subclass of EPCISEvent)

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.

Field	Type	Description
`eventTime` `recordTime`	(Inherited from `EPCISEvent`; see Section 7.2.8)	
`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 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 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.

7.2.12 TransactionEvent (subclass of EPCISEvent)

The event type `TransactionEvent` describes the association or disassociation of physical 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 EPCs 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 EPCs named 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 EPCs are added to an existing transaction(s).

`Action` value	Meaning
`OBSERVE`	The EPCs 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 EPCs named in the event have been disassociated from the business transaction(s) during this event. This includes situations where a subset of EPCs are disassociated from the business transaction(s), as well as when the entire business transaction(s) has ended. As a convenience, the list of EPCs may be omitted from the `TransactionEvent`, which means that *all* EPCs have been disassociated.

1280

1281 A `TransactionEvent` has the following fields:

Field	Type	Description
`eventTime` `recordTime`	(Inherited from `EPCISEvent`; see Section 7.2.8)	
`bizTransactionList`	Unordered list of one or more `BusinessTransaction` instances	The business transaction(s).
`parentID`	`URI`	(Optional) The identifier of the parent of the EPCs given in `epcList`. When the parent identifier is an EPC, this field SHALL contain the "pure identity" URI for the EPC as specified in [TDS1.3], Section 4.1. See also the note following the table.

Field	Type	Description
epcList	List<EPC>	An unordered list of the EPCs of the objects associated with the business transaction. Each element of the list SHALL be a URI [RFC2396] denoting the unique identity for a physical object. When the unique identity is an Electronic Product Code, the list element SHALL be the "pure identity" URI for the contained EPC as specified in [TDS1.3], Section 4.1. Implementations MAY accept URI-formatted identifiers other than EPCs. The epcList MAY be empty if action is DELETE, indicating that all the EPCs 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 EPCs, presumed to hold 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.

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. All the non-EPC information fields (*`bizTransactionList`*,* `bizStep`*,* `bizLocation`*, etc) apply equally and uniformly to all EPCs specified, whether or not the EPCs are specified in just the* `epcList` *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 specific EPCs. 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:

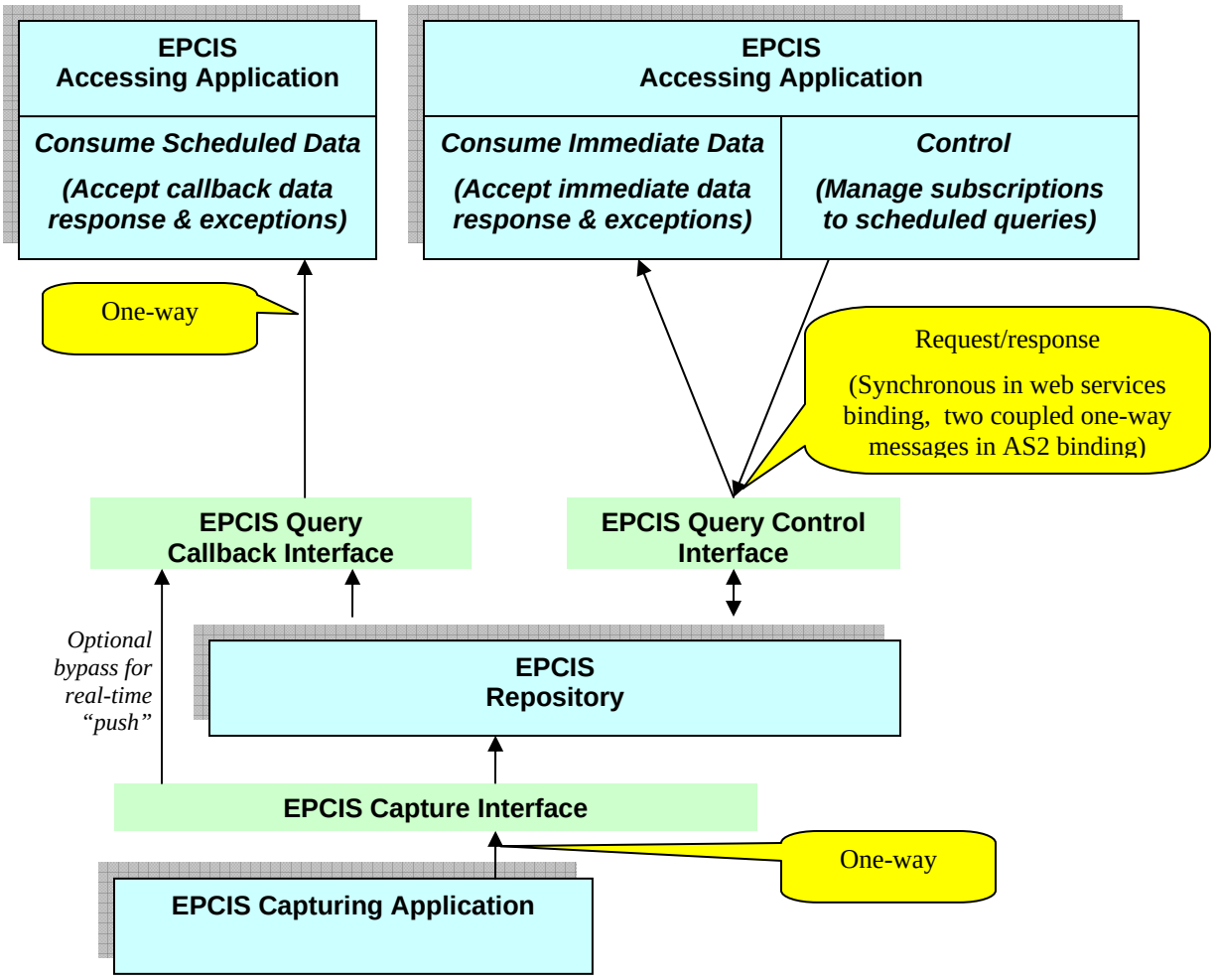
- An event described by `bizStep` (and any other fields) took place involving the business transactions enumerated in `bizTransactionList`, the EPCs in `epcList`, and containing entity `parentID` (if specified), at `eventTime` and location `readPoint`.
- (If `action` is `ADD`) The EPCs in `epcList` and containing entity `parentID` (if specified) were associated to the business transactions enumerated in `bizTransactionList`.
- (If `action` is `DELETE` and `epcList` is non-empty) The EPCs in `epcList` and containing entity `parentID` (if specified) were disassociated from the business transactions enumerated in `bizTransactionList`.
- (If `action` is `DELETE`, `epcList` is empty, and `parentID` is omitted) All EPCs have been disassociated from the business transactions enumerated in `bizTransactionList`.

Prospective semantics:

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

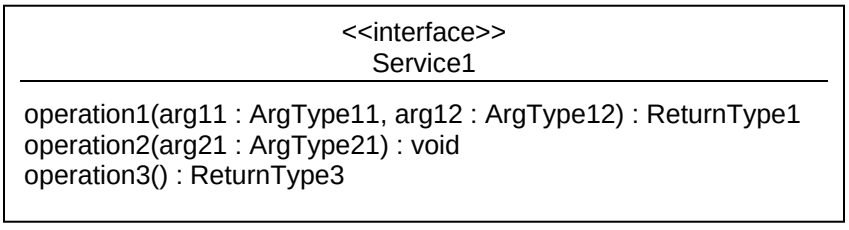
8 Service Layer

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):



1342

1343 In the subsections below, services are specified using UML class diagram notation.
1344 UML class diagrams used for this purpose may contain interfaces having operations, but
1345 not fields or associations. Here is an example:



1346

1347 This diagram shows a service definition for `Service1`, which provides three operations.
1348 `Operation1` takes two arguments, `arg11` and `arg12`, having types `ArgType11` and
1349 `ArgType12`, respectively, and returns a value of type `ReturnType1`. `Operation2`
1350 takes one argument but does not return a result. `Operation3` does not take any
1351 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 EPCglobal 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.

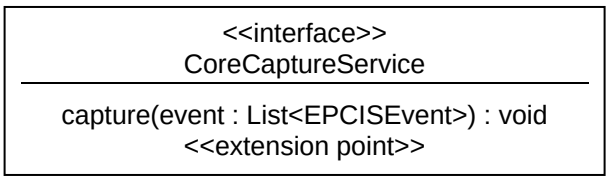
8.1.1 Authentication and Authorization

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 honor 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 7.2.3), 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 "pull" and "push," respectively.

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

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

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

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 EPCglobal. 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 1.0 of the EPCIS specification, the implementation SHALL return the string `1.0`.
`getVendorVersion`	Returns a string that identifies what vendor extensions this implementation provides. The possible values of this string and their meanings are vendor-defined, except that the empty string SHALL indicate that the implementation implements only standard functionality with no vendor extensions. When an implementation chooses to return a non-empty string, the value returned SHALL be a URI where the vendor is the owning authority. For example, this may be an HTTP URL whose authority portion is a domain name owned by the vendor, a URN having a URN namespace identifier issued to the vendor by IANA, an OID URN whose initial path is a Private Enterprise Number assigned to the vendor, etc.

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

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

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

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

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

8.2.5.1 Subscription Controls

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

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

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

Argument	Type	Description
`schedule`	`QuerySchedule`	(Optional) Defines the periodic schedule on which the query is to be executed. See 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 `Element`s of the query schedule field.
- A value extracted from the time value matches an `Element` of a query schedule field if
  - the `Element` is a `Number` and the value extracted from the time value is equal to the `Number`; or
  - the `Element` is a `Range` and the value extracted from the time value is greater than or equal to the first `Number` in the `Range` and less than or equal to the second `Number` in the `Range`.

See examples following the table below.

An EPCIS implementation SHALL interpret the `QuerySchedule` as a client's statement of when it would like the query to be executed, and SHOULD make reasonable efforts to adhere to that schedule. An EPCIS implementation MAY, however, deviate from the requested schedule according to its own policies regarding server load, 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 honor 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 honor that request precisely, but can honor 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.

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

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`).

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

8.2.6 Error Conditions

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

Exception Name	Meaning
`SecurityException`	The operation was not permitted due to an access control violation or other security concern. This includes the case where the service wishes to deny 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.

Exception Name	Meaning
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.
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`.

Exception Name	Meaning
`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.

1735

1736 The exceptions that may be thrown by each method of the EPCIS Query Control
1737 Interface are indicated in the table below:

EPCIS Method	Exceptions

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

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

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

EPCIS Method	Exceptions
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 theEPCIS 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 1.0

In EPCIS 1.0, no query language is provided by which a client may express an arbitrary query for data. Instead, an EPCIS 1.0 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:

Parameter Name	Parameter Value Type	Required	Meaning
`eventType`	List of String	No	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`, or `TransactionEvent`. 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.

Parameter Name	Parameter Value Type	Required	Meaning
GE_eventTime	Time	No	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	No	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	No	If provided, only events with recordTime greater than or equal to the specified value will be returned. The automatic limitation based on event record time (Section 8.2.5.2) may implicitly provide a constraint similar to this parameter. If omitted, events are included regardless of their recordTime, other than automatic limitation based on event record time (Section 8.2.5.2).
LT_recordTime	Time	No	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).

Parameter Name	Parameter Value Type	Required	Meaning
`EQ_action`	List of String	No	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	No	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	No	Like the `EQ_bizStep` parameter, but for the `disposition` field.
`EQ_readPoint`	List of String	No	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	Required	Meaning
`WD_readPoint`	List of String	No	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	No	Like the `EQ_readPoint` parameter, but for the `bizLocation` field.
`WD_bizLocation`	List of String	No	Like the `WD_readPoint` parameter, but for the `bizLocation` field.
`EQ_bizTransaction_`*`type`*	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 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.

Parameter Name	Parameter Value Type	Required	Meaning
MATCH_epc	List of String	No	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. Each element of the parameter list may be a pure identity pattern as specified in [TDS1.3], 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 [TDS1.3, Section 6]. If the element is any other URI, it is matched against event field values by testing string equality. 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	Required	Meaning
MATCH_parentID	List of String	No	Like MATCH_epc, but applies to the parentID field of AggregationEvent, the parentID field of TransactionEvent, and extension event types that extend either AggregationEvent or TransactionEvent. Each element of the parameter list may be a pure identity pattern as specified in [TDS1.3], 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 [TDS1.3, Section 6]. If the element is any other URI, it is matched against event field values by testing string equality.
MATCH_anyEPC	List of String	No	If this parameter is specified, the result will only include events that (a) have an epcList field, a childEPCs field, or a parentID field (that is, ObjectEvent, AggregationEvent, TransactionEvent or extension event types that extend one of those three); and where (b) the parentID field or 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. Each element of the parameter list may be a pure identity pattern as specified in [TDS1.3], 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 [TDS1.3, Section 6]. If the element is any other URI, it is matched against event field values by testing string equality.

Parameter Name	Parameter Value Type	Required	Meaning
`MATCH_epcClass`	List of String	No	Like `MATCH_epc`, but applies to the `epcClass` field of `QuantityEvent`s or extension event types that extend `QuantityEvent`. The definition of a "match" for the purposes of this query parameter is as follows. Let *P* be one of the patterns specified in the value for this parameter, and let *C* be the value of the `epcClass` field of a QuantityEvent being considered for inclusion in the result. Then the QuantityEvent is included if each component *Pi* of *P* matches the corresponding component *Ci* of *C*, where "matches" is as defined in [TDS1.3, Section 6]. *Explanation (non-normative): The difference between MATCH_epcClass and MATCH_epc is that for MATCH_epcClass the value in the event (the epcClass field of the QuantityEvent) may itself be a pattern, as specified in Section 7.2.7). This means that the value in the event may contain a '*' component. The above specification says that a '*' in the QuantityEvent is only matched by a '*' in the query parameter. For example, if the epcClass field of a QuantityEvent 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.*

Parameter Name	Parameter Value Type	Required	Meaning
`EQ_quantity`	Int	No	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	No	Like `EQ_quantity`, but includes events whose `quantity` field is greater than the specified parameter.
`GE_quantity`	Int	No	Like `EQ_quantity`, but includes events whose `quantity` field is greater than or equal to the specified parameter.
`LT_quantity`	Int	No	Like `EQ_quantity`, but includes events whose `quantity` field is less than the specified parameter.
`LE_quantity`	Int	No	Like `EQ_quantity`, but includes events whose `quantity` field is less than or equal to the specified parameter.
`EQ_`*`fieldname`*	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 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.

Parameter Name	Parameter Value Type	Required	Meaning
EQ_*fieldname*	Int Float Time	No	Like EQ_*fieldname* as described above, but may be applied to a field of type Int, Float, or Time. The result will include events that (a) have a field named *fieldname*; and where (b) the type of the field matches the type of this parameter (Int, Float, or Time); and where (c) the value of the field is equal to the specified value. *Fieldname* is constructed as for EQ_*fieldname*.
GT_*fieldname*	Int Float Time	No	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	No	Analogous to GT_*fieldname*
EXISTS_*fieldname*	Void	No	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	Required	Meaning
HASATTR_*fieldname*	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 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	Required	Meaning
EQATTR_*fieldname*_*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 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	Required	Meaning
`orderBy`	String	No	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`, `quantity`, 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.
`orderDirection`	String	No	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 descencing order); if not, the implementation SHALL raise a `QueryParameterException`. If omitted, defaults to `DESC`.
`eventCountLimit`	Int	No	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

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

Parameter Name	Parameter Value Type	Required	Meaning
maxEventCount	Int	No	If specified, at most this many events will be included in the query result. If the query would otherwise return more than this number of events, a QueryTooLargeException SHALL be raised instead of a normal query result. This parameter and eventCountLimit are mutually exclusive; if both are specified, a QueryParameterException SHALL be raised. If this parameter is omitted, any number of events may be included in the query result. Note, however, that the EPCIS implementation is free to raise a QueryTooLargeException regardless of the setting of this parameter (see Section 8.2.3).

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"))

8.2.7.2 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 specifed in the table at the end of Section 7.2. If omitted, all vocabularies are considered.
`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.

Parameter Name	Parameter Value Type	Required	Meaning
`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).

1806

1807 As the descriptions above suggest, if multiple parameters are specified a vocabulary
1808 element must satisfy all criteria in order to be included in the result set. In other words, if
1809 each parameter is considered to be a predicate, all such predicates are implicitly
1810 conjoined as though by an AND operator. For example, if a given call to `poll` specifies
1811 a value for both the `WD_name` and `HASATTR` parameters, then a vocabulary element
1812 must be a descendant of the specified element AND possess one of the specified
1813 attributes in order to be included in the result.

1814 On the other hand, for those parameters whose value is a list, a vocabulary element must
1815 match *at least one* of the elements of the list in order to be included in the result set. In
1816 other words, if each element of the list is considered to be a predicate, all such predicates
1817 for a given list are implicitly disjoined as though by an OR operator. For example, if the
1818 value of the `EQATTR_sample` parameter is a two element list ("`s1`", "`s2`"), then a
1819 vocabulary element is included if it has a `sample` attribute whose value is equal to `s1`
1820 OR equal to `s2`.

1821 As another example, if the value of the `EQ_name` parameter is a two element list
1822 ("`ve1`", "`ve2`") and the `EQATTR_sample` parameter is a two element list ("`s1`",
1823 "`s2`"), then the effect is to include events satisfying the following predicate:

1824 ((`name` = "ve1" `OR` `name` = "ve2")
1825 `AND` (`sample` = "s1" `OR` `sample` = "s2"))

1826 where `name` informally refers to the name of the vocabulary element and `sample`
1827 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 EPCglobal 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 extension, and for extension by EPCglobal 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 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-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 EPCglobal Version 1.0 schema will conform both to the EPCglobal 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 type-checking of the extension elements and attributes, but when the document is parsed using the vendor-specific schema the extensions will be type-checked. Similarly, a document containing new features introduced in a hypothetical EPCglobal Version 1.1 schema will conform both to the EPCglobal Version 1.0 schema and to the EPCglobal Version 1.1 schema, but type checking 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 attributes, either in subsequent versions of the standard schema or in vendor-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-specific schema.

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

- Vendor-specific attributes may be added to any type in which `<<extension point>>` occurs. Vendor-specific attributes SHALL NOT be in the EPCglobal EPCIS namespace (`urn:epcglobal:epcis:xsd:1`). Vendor-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-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, 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. Declarations of vendor-specific attributes SHALL specify `use="optional"`.

- Vendor-specific elements may be added to any type in which `<<extension point>>` occurs. Vendor-specific elements SHALL NOT be in the EPCglobal EPCIS namespace (`urn:epcglobal:epcis:xsd:1`). Vendor-specific elements SHALL be in a namespace whose namespace URI has the vendor as the owning authority (as described above). (In schema parlance, this means that all vendor-specific elements must have `qualified` as their form.)

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

Explanation (non-normative): Because vendor-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-specific elements. Moving the `<xsd:any …>` into the vendor's schema avoids this problem, because `##other` in that schema means "match an element that has a namespace other than the vendor'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's schema does not match standard EPCIS elements, either.

The rules for adding attributes or elements to future versions of the EPCglobal 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, 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); 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`.

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`
`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`

Document Type and Context	Value for `DocumentIdentification/Type`
`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 0
`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="1.0">
        <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"/>
        <!-- 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="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="EPCISBodyType">
                <xsd:annotation>
                        <xsd:documentation xml:lang="en">
                        specific body that contains EPCIS related Events.
                        </xsd:documentation>
                </xsd:annotation>
                <xsd:sequence>
                        <xsd:element name="EventList" type="epcis:EventListType"
minOccurs="0"/>
                        <xsd:element name="extension" type="epcis:EPCISBodyExtensionType"
minOccurs="0"/>
                        <xsd:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
                </xsd:sequence>
                <xsd:anyAttribute processContents="lax"/>
        </xsd:complexType>
        <!-- EPCIS CORE ELEMENT TYPES -->
        <xsd:complexType name="EventListType">
                <!-- Note: the use of "unbounded" in both the xsd:choice element
                     and the enclosed xsd:element elements is, strictly speaking,
                     redundant. However, this was found to avoid problems with
                     certain XML processing tools, and so is retained here.
                 -->
                <xsd:choice minOccurs="0" maxOccurs="unbounded">
                        <xsd:element name="ObjectEvent" type="epcis:ObjectEventType"
minOccurs="0" maxOccurs="unbounded"/>
                        <xsd:element name="AggregationEvent"
type="epcis:AggregationEventType" minOccurs="0" maxOccurs="unbounded"/>
                        <xsd:element name="QuantityEvent" type="epcis:QuantityEventType"
minOccurs="0" maxOccurs="unbounded"/>
                        <xsd:element name="TransactionEvent"
type="epcis:TransactionEventType" minOccurs="0" maxOccurs="unbounded"/>
                        <xsd:element name="extension"
type="epcis:EPCISEventListExtensionType"/>
                        <xsd:any namespace="##other" processContents="lax"/>
```

```
2180                </xsd:choice>
2181          </xsd:complexType>
2182          <xsd:complexType name="EPCListType">
2183                <xsd:sequence>
2184                      <xsd:element name="epc" type="epcglobal:EPC" minOccurs="0"
2185    maxOccurs="unbounded"/>
2186                </xsd:sequence>
2187          </xsd:complexType>
2188          <xsd:simpleType name="ActionType">
2189                <xsd:restriction base="xsd:string">
2190                      <xsd:enumeration value="ADD"/>
2191                      <xsd:enumeration value="OBSERVE"/>
2192                      <xsd:enumeration value="DELETE"/>
2193                </xsd:restriction>
2194          </xsd:simpleType>
2195          <xsd:simpleType name="ParentIDType">
2196                <xsd:restriction base="xsd:anyURI"/>
2197          </xsd:simpleType>
2198          <!-- Standard Vocabulary -->
2199          <xsd:simpleType name="BusinessStepIDType">
2200                <xsd:restriction base="xsd:anyURI"/>
2201          </xsd:simpleType>
2202          <!-- Standard Vocabulary -->
2203          <xsd:simpleType name="DispositionIDType">
2204                <xsd:restriction base="xsd:anyURI"/>
2205          </xsd:simpleType>
2206          <!-- User Vocabulary -->
2207          <xsd:simpleType name="EPCClassType">
2208                <xsd:restriction base="xsd:anyURI"/>
2209          </xsd:simpleType>
2210          <!-- User Vocabulary -->
2211          <xsd:simpleType name="ReadPointIDType">
2212                <xsd:restriction base="xsd:anyURI"/>
2213          </xsd:simpleType>
2214          <xsd:complexType name="ReadPointType">
2215            <xsd:sequence>
2216              <xsd:element name="id" type="epcis:ReadPointIDType"/>
2217              <xsd:element name="extension" type="epcis:ReadPointExtensionType"
2218    minOccurs="0"/>
2219              <!-- The wildcard below provides the extension mechanism described in Section
2220    9.4 -->
2221              <xsd:any namespace="##other" processContents="lax" minOccurs="0"
2222    maxOccurs="unbounded"/>
2223            </xsd:sequence>
2224          </xsd:complexType>
2225          <xsd:complexType name="ReadPointExtensionType">
2226            <xsd:sequence>
2227              <xsd:any namespace="##local" processContents="lax" maxOccurs="unbounded"/>
2228            </xsd:sequence>
2229            <xsd:anyAttribute processContents="lax"/>
2230          </xsd:complexType>
2231          <!-- User Vocabulary -->
2232          <xsd:simpleType name="BusinessLocationIDType">
2233                <xsd:restriction base="xsd:anyURI"/>
2234          </xsd:simpleType>
2235          <xsd:complexType name="BusinessLocationType">
2236            <xsd:sequence>
2237              <xsd:element name="id" type="epcis:BusinessLocationIDType"/>
2238              <xsd:element name="extension" type="epcis:BusinessLocationExtensionType"
2239    minOccurs="0"/>
2240              <!-- The wildcard below provides the extension mechanism described in Section
2241    9.4 -->
2242              <xsd:any namespace="##other" processContents="lax" minOccurs="0"
2243    maxOccurs="unbounded"/>
2244            </xsd:sequence>
2245          </xsd:complexType>
2246          <xsd:complexType name="BusinessLocationExtensionType">
2247            <xsd:sequence>
2248              <xsd:any namespace="##local" processContents="lax" maxOccurs="unbounded"/>
2249            </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>
        <!-- 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).

        In this approach, an optional <extension> element is defined
        for each extensible element type, where an <extension> element
        may contain future elements defined in the target namespace.

        In addition to the optional <extension> element, extensible element
        types are declared with a final xsd:any wildcard to accommodate
        future elements defined by third parties (as denoted by the ##other
        namespace).

        Finally, the xsd:anyAttribute facility is used to allow arbitrary
        attributes to be added to extensible element types. -->
        <xsd:complexType name="EPCISEventType" abstract="true">
               <xsd:annotation>
                      <xsd:documentation xml:lang="en">
                      base type for all EPCIS events.
                      </xsd:documentation>
               </xsd:annotation>
               <xsd:sequence>
                      <xsd:element name="eventTime" type="xsd:dateTime"/>
                      <xsd:element name="recordTime" type="xsd:dateTime" minOccurs="0"/>
                      <xsd:element name="eventTimeZoneOffset" type="xsd:string"/>
                      <xsd:element name="baseExtension"
type="epcis:EPCISEventExtensionType" minOccurs="0"/>
               </xsd:sequence>
               <xsd:anyAttribute processContents="lax"/>
        </xsd:complexType>
        <xsd:complexType name="ObjectEventType">
               <xsd:annotation>
                      <xsd:documentation xml:lang="en">
                      Object Event captures information about an event pertaining to one
or more
                      objects identified by EPCs.
               </xsd:documentation>
               </xsd:annotation>
               <xsd:complexContent>
                      <xsd:extension base="epcis:EPCISEventType">
                             <xsd:sequence>
                                    <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="bizTransactionList"
type="epcis:BusinessTransactionListType" minOccurs="0"/>
                                        <xsd:element name="extension"
type="epcis:ObjectEventExtensionType" 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="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>
        <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="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>
        <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="EPCISHeaderExtensionType">
                <xsd:sequence>
                        <xsd:any namespace="##local" processContents="lax"
maxOccurs="unbounded"/>
                </xsd:sequence>
                <xsd:anyAttribute processContents="lax"/>
        </xsd:complexType>
        <xsd:complexType name="EPCISBodyExtensionType">
                <xsd:sequence>
                        <xsd:any namespace="##local" processContents="lax"
maxOccurs="unbounded"/>
```

```
                </xsd:sequence>
                <xsd:anyAttribute processContents="lax"/>
        </xsd:complexType>
        <xsd:complexType name="EPCISEventListExtensionType">
                <xsd:sequence>
                        <xsd:any namespace="##local" processContents="lax"
maxOccurs="unbounded"/>
                </xsd:sequence>
                <xsd:anyAttribute processContents="lax"/>
        </xsd:complexType>
        <xsd:complexType name="EPCISEventExtensionType">
                <xsd:sequence>
                        <xsd:any namespace="##local" processContents="lax"
maxOccurs="unbounded"/>
                </xsd:sequence>
                <xsd:anyAttribute processContents="lax"/>
        </xsd:complexType>
        <xsd:complexType name="ObjectEventExtensionType">
                <xsd:sequence>
                        <xsd:any namespace="##local" processContents="lax"
maxOccurs="unbounded"/>
                </xsd:sequence>
                <xsd:anyAttribute processContents="lax"/>
        </xsd:complexType>
        <xsd:complexType name="AggregationEventExtensionType">
                <xsd:sequence>
                        <xsd:any namespace="##local" processContents="lax"
maxOccurs="unbounded"/>
                </xsd:sequence>
                <xsd:anyAttribute processContents="lax"/>
        </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="TransactionEventExtensionType">
                <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 – Example (non-normative)

Here is an example `EPCISDocument` containing two `ObjectEvents`, rendered into XML [XML1.0]:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<epcis:EPCISDocument
    xmlns:epcis="urn:epcglobal:epcis:xsd:1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    creationDate="2005-07-11T11:30:47.0Z"
    schemaVersion="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>
        </epcList>
```

```
          <action>OBSERVE</action>
          <bizStep>urn:epcglobal:epcis:bizstep:fmcg:shipped</bizStep>
          <disposition>urn:epcglobal:epcis:disp:fmcg:unknown</disposition>
          <readPoint>
             <id>urn:epc:id:sgln:0614141.07346.1234</id>
          </readPoint>
          <bizLocation>
             <id>urn:epcglobal:fmcg:loc:0614141073467.A23-49</id>
          </bizLocation>
          <bizTransactionList>
              <bizTransaction
type="urn:epcglobal:fmcg: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:epcis:bizstep:fmcg:received</bizStep>
          <disposition>urn:epcglobal:epcis:disp:fmcg:processing</disposition>
          <readPoint>
             <id>urn:epcglobal:fmcg:loc:0614141073467.RP-1529</id>
          </readPoint>
          <bizLocation>
             <id>urn:epcglobal:fmcg:loc:0614141073467.A23-49-shelf1234</id>
          </bizLocation>
          <bizTransactionList>
              <bizTransaction
type="urn:epcglobal:fmcg:btt:po">http://transaction.acme.com/po/12345678</bizTransaction>
              <bizTransaction
type="urn:epcglobal:fmcg:btt:asn">http://transaction.acme.com/asn/1152</bizTransaction>
          </bizTransactionList>
        </ObjectEvent>
      </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 0
`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_0.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_0.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"
   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="urn:epcglobal:fmcg:mda:slt:retail"/>
            <attribute id="urn:epcglobal:fmcg:mda:latitude">+18.0000</attribute>
            <attribute id="urn:epcglobal:fmcg:mda:longitude">-70.0000</attribute>
            <attribute id="urn:epcglobal:fmcg:mda:address">
              <sample:Address xmlns:sample="http://sample.com/ComplexTypeExample">
                <Street>100 Nowhere Street</Street>
                <City>Fancy</City>
                <State>FiftyOne</State>
                <Zip>99999</Zip>
```

```
              </sample:Address>
            </attribute>
            <children>
              <id>urn:epcglobal:fmcg:ssl:0037000.00729.201</id>
              <id>urn:epcglobal:fmcg:ssl:0037000.00729.202</id>
              <id>urn:epcglobal:fmcg:ssl:0037000.00729.203</id>
            </children>
          </VocabularyElement>
          <VocabularyElement id="urn:epcglobal:fmcg:ssl:0037000.00729.201">
            <attribute id="urn:epcglobal:fmcg:mda:sslt:201"/>
          </VocabularyElement>
          <VocabularyElement id="urn:epcglobal:fmcg:ssl:0037000.00729.202">
            <attribute id="urn:epcglobal:fmcg:mda:sslt:202"/>
            <children>
              <id>urn:epcglobal:fmcg:ssl:0037000.00729.202,402</id>
            </children>
          </VocabularyElement>
          <VocabularyElement id="urn:epcglobal:fmcg:ssl:0037000.00729.202,402">
            <attribute id="urn:epcglobal:fmcg:mda:sslt:202"/>
            <attribute id="urn:epcglobal:fmcg:mda:sslta:402"/>
          </VocabularyElement>
        </VocabularyElementList>
      </Vocabulary>
      <Vocabulary type="urn:epcglobal:epcis:vtype:ReadPoint">
        <VocabularyElementList>
          <VocabularyElement id="urn:epcglobal:fmcg:ssl:0037000.00729.201">
            <attribute
id="urn:epcglobal:epcis:mda:site">urn:epc:id:sgln:0037000.00729.0</attribute>
            <attribute id="urn:epcglobal:fmcg:mda:sslt:201"/>
          </VocabularyElement>
          <VocabularyElement id="urn:epcglobal:fmcg:ssl:0037000.00729.202">
            <attribute
id="urn:epcglobal:epcis:mda:site">urn:epc:id:sgln:0037000.00729.0</attribute>
            <attribute id="urn:epcglobal:fmcg:mda:sslt:202"/>
          </VocabularyElement>
          <VocabularyElement id="urn:epcglobal:fmcg:ssl:0037000.00729.203">
            <attribute
id="urn:epcglobal:epcis:mda:site">urn:epc:id:sgln:0037000.00729.0</attribute>
            <attribute id="urn:epcglobal:fmcg:mda:sslt:203"/>
          </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 Messsage 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. Successful delivery of this message to 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 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. Successful delivery of this message to 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 0
`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_0.xsd	Section 9.5
`urn:epcglobal:epcis-masterdata:xsd:1`	EPCglobal-epcis-masterdata-1_0.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_0.xsd"/>
  <xsd:import namespace="urn:epcglobal:epcis-masterdata:xsd:1"
schemaLocation="./EPCglobal-epcis-masterdata-1_0.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"/>
            <xsd:element name="extension" type="epcisq:EPCISQueryDocumentExtensionType"
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="EPCISQueryDocumentExtensionType">
               <xsd:sequence>
                     <xsd:any namespace="##local" processContents="lax"
maxOccurs="unbounded"/>
               </xsd:sequence>
               <xsd:anyAttribute processContents="lax"/>
        </xsd:complexType>

```

```
    <xsd:complexType name="EPCISQueryBodyType">
       <xsd:choice>
          <xsd:element ref="epcisq:GetQueryNames"/>
          <xsd:element ref="epcisq:GetQueryNamesResult"/>
          <xsd:element ref="epcisq:Subscribe"/>
          <xsd:element ref="epcisq:SubscribeResult"/>
          <xsd:element ref="epcisq:Unsubscribe"/>
          <xsd:element ref="epcisq:UnsubscribeResult"/>
          <xsd:element ref="epcisq:GetSubscriptionIDs"/>
          <xsd:element ref="epcisq:GetSubscriptionIDsResult"/>
          <xsd:element ref="epcisq:Poll"/>
          <xsd:element ref="epcisq:GetStandardVersion"/>
          <xsd:element ref="epcisq:GetStandardVersionResult"/>
          <xsd:element ref="epcisq:GetVendorVersion"/>
          <xsd:element ref="epcisq:GetVendorVersionResult"/>
          <xsd:element ref="epcisq:DuplicateNameException"/>
        <!-- queryValidationException unimplemented in EPCIS 1.0
          <xsd:element ref="epcisq:QueryValidationException"/>
        -->
          <xsd:element ref="epcisq:InvalidURIException"/>
          <xsd:element ref="epcisq:NoSuchNameException"/>
          <xsd:element ref="epcisq:NoSuchSubscriptionException"/>
          <xsd:element ref="epcisq:DuplicateSubscriptionException"/>
          <xsd:element ref="epcisq:QueryParameterException"/>
          <xsd:element ref="epcisq:QueryTooLargeException"/>
          <xsd:element ref="epcisq:QueryTooComplexException"/>
          <xsd:element ref="epcisq:SubscriptionControlsException"/>
          <xsd:element ref="epcisq:SubscribeNotPermittedException"/>
          <xsd:element ref="epcisq:SecurityException"/>
          <xsd:element ref="epcisq:ValidationException"/>
          <xsd:element ref="epcisq:ImplementationException"/>
          <xsd:element ref="epcisq:QueryResults"/>
       </xsd:choice>
    </xsd:complexType>

    <!-- EPCISSERVICE MESSAGE WRAPPERS -->

    <xsd:element name="GetQueryNames" type="epcisq:EmptyParms"/>
    <xsd:element name="GetQueryNamesResult" type="epcisq:ArrayOfString"/>

    <xsd:element name="Subscribe" type="epcisq:Subscribe"/>
    <xsd:complexType name="Subscribe">
       <xsd:sequence>
          <xsd:element name="queryName" type="xsd:string"/>
          <xsd:element name="params" type="epcisq:QueryParams"/>
          <xsd:element name="dest" type="xsd:anyURI"/>
          <xsd:element name="controls" type="epcisq:SubscriptionControls"/>
          <xsd:element name="subscriptionID" type="xsd:string"/>
       </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="SubscribeResult" type="epcisq:VoidHolder"/>

    <xsd:element name="Unsubscribe" type="epcisq:Unsubscribe"/>
    <xsd:complexType name="Unsubscribe">
       <xsd:sequence>
          <xsd:element name="subscriptionID" type="xsd:string"/>
       </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="UnsubscribeResult" type="epcisq:VoidHolder"/>

    <xsd:element name="GetSubscriptionIDs" type="epcisq:GetSubscriptionIDs"/>
    <xsd:complexType name="GetSubscriptionIDs">
       <xsd:sequence>
          <xsd:element name="queryName" type="xsd:string"/>
       </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="GetSubscriptionIDsResult" type="epcisq:ArrayOfString"/>

    <xsd:element name="Poll" type="epcisq:Poll"/>
    <xsd:complexType name="Poll">
```

```
      <xsd:sequence>
         <xsd:element name="queryName" type="xsd:string"/>
         <xsd:element name="params" type="epcisq:QueryParams"/>
      </xsd:sequence>
   </xsd:complexType>
   <!-- The response from a Poll method is the QueryResults element, defined below.
        The QueryResults element is also used to deliver standing query results
        through the Query Callback Interface -->

   <xsd:element name="GetStandardVersion" type="epcisq:EmptyParms"/>
   <xsd:element name="GetStandardVersionResult" type="xsd:string"/>

   <xsd:element name="GetVendorVersion" type="epcisq:EmptyParms"/>
   <xsd:element name="GetVendorVersionResult" type="xsd:string"/>

   <xsd:element name="VoidHolder" type="epcisq:VoidHolder"/>
   <xsd:complexType name="VoidHolder">
      <xsd:sequence>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:complexType name="EmptyParms"/>

   <xsd:complexType name="ArrayOfString">
      <xsd:sequence>
         <xsd:element name="string" type="xsd:string" minOccurs="0"
maxOccurs="unbounded"/>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:complexType name="SubscriptionControls">
     <xsd:sequence>
       <xsd:element name="schedule" type="epcisq:QuerySchedule" minOccurs="0"/>
       <xsd:element name="trigger" type="xsd:anyURI" minOccurs="0"/>
       <xsd:element name="initialRecordTime" type="xsd:dateTime" minOccurs="0"/>
       <xsd:element name="reportIfEmpty" type="xsd:boolean"/>
       <xsd:element name="extension" type="epcisq:SubscriptionControlsExtensionType"
minOccurs="0"/>
       <xsd:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
     </xsd:sequence>
   </xsd:complexType>

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

   <xsd:complexType name="QuerySchedule">
     <xsd:sequence>
        <xsd:element name="second" type="xsd:string" minOccurs="0"/>
        <xsd:element name="minute" type="xsd:string" minOccurs="0"/>
        <xsd:element name="hour" type="xsd:string" minOccurs="0"/>
        <xsd:element name="dayOfMonth" type="xsd:string" minOccurs="0"/>
        <xsd:element name="month" type="xsd:string" minOccurs="0"/>
        <xsd:element name="dayOfWeek" type="xsd:string" minOccurs="0"/>
       <xsd:element name="extension" type="epcisq:QueryScheduleExtensionType"
minOccurs="0"/>
       <xsd:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
     </xsd:sequence>
   </xsd:complexType>

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

```
    </xsd:complexType>

    <xsd:complexType name="QueryParams">
      <xsd:sequence>
        <xsd:element name="param" type="epcisq:QueryParam" minOccurs="0"
maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="QueryParam">
      <xsd:sequence>
        <xsd:element name="name" type="xsd:string"/>
        <!-- See note in EPCIS spec text regarding the value for this element -->
        <xsd:element name="value" type="xsd:anyType"/>
      </xsd:sequence>
    </xsd:complexType>

    <xsd:element name="QueryResults" type="epcisq:QueryResults"/>
    <xsd:complexType name="QueryResults">
      <xsd:sequence>
        <xsd:element name="queryName" type="xsd:string"/>
        <xsd:element name="subscriptionID" type="xsd:string" minOccurs="0"/>
        <xsd:element name="resultsBody" type="epcisq:QueryResultsBody"/>
        <xsd:element name="extension" type="epcisq:QueryResultsExtensionType"
minOccurs="0"/>
        <xsd:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>

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

    <xsd:complexType name="QueryResultsBody">
      <xsd:choice>
        <xsd:element name="EventList" type="epcis:EventListType"/>
        <xsd:element name="VocabularyList" type="epcismd:VocabularyListType"/>
      </xsd:choice>
    </xsd:complexType>

    <!-- EPCIS EXCEPTIONS -->

    <xsd:element name="EPCISException" type="epcisq:EPCISException"/>
    <xsd:complexType name="EPCISException">
       <xsd:sequence>
          <xsd:element name="reason" type="xsd:string"/>
       </xsd:sequence>
    </xsd:complexType>

    <xsd:element name="DuplicateNameException" type="epcisq:DuplicateNameException"/>
    <xsd:complexType name="DuplicateNameException">
       <xsd:complexContent>
          <xsd:extension base="epcisq:EPCISException">
             <xsd:sequence/>
          </xsd:extension>
       </xsd:complexContent>
    </xsd:complexType>

    <!-- QueryValidationException not implemented in EPCIS 1.0
    <xsd:element name="QueryValidationException" type="epcisq:QueryValidationException"/>
    <xsd:complexType name="QueryValidationException">
       <xsd:complexContent>
          <xsd:extension base="epcisq:EPCISException">
             <xsd:sequence/>
          </xsd:extension>
       </xsd:complexContent>
```

```
    </xsd:complexType>
    -->

    <xsd:element name="InvalidURIException" type="epcisq:InvalidURIException"/>
    <xsd:complexType name="InvalidURIException">
       <xsd:complexContent>
          <xsd:extension base="epcisq:EPCISException">
             <xsd:sequence/>
          </xsd:extension>
       </xsd:complexContent>
    </xsd:complexType>

    <xsd:element name="NoSuchNameException" type="epcisq:NoSuchNameException"/>
    <xsd:complexType name="NoSuchNameException">
       <xsd:complexContent>
          <xsd:extension base="epcisq:EPCISException">
             <xsd:sequence/>
          </xsd:extension>
       </xsd:complexContent>
    </xsd:complexType>

    <xsd:element name="NoSuchSubscriptionException"
  type="epcisq:NoSuchSubscriptionException"/>
    <xsd:complexType name="NoSuchSubscriptionException">
       <xsd:complexContent>
          <xsd:extension base="epcisq:EPCISException">
             <xsd:sequence/>
          </xsd:extension>
       </xsd:complexContent>
    </xsd:complexType>

    <xsd:element name="DuplicateSubscriptionException"
  type="epcisq:DuplicateSubscriptionException"/>
    <xsd:complexType name="DuplicateSubscriptionException">
       <xsd:complexContent>
          <xsd:extension base="epcisq:EPCISException">
             <xsd:sequence/>
          </xsd:extension>
       </xsd:complexContent>
    </xsd:complexType>

    <xsd:element name="QueryParameterException" type="epcisq:QueryParameterException"/>
    <xsd:complexType name="QueryParameterException">
       <xsd:complexContent>
          <xsd:extension base="epcisq:EPCISException">
             <xsd:sequence/>
          </xsd:extension>
       </xsd:complexContent>
    </xsd:complexType>

    <xsd:element name="QueryTooLargeException" type="epcisq:QueryTooLargeException"/>
    <xsd:complexType name="QueryTooLargeException">
       <xsd:complexContent>
          <xsd:extension base="epcisq:EPCISException">
             <xsd:sequence>
               <xsd:element name="queryName" type="xsd:string" minOccurs="0"/>
               <xsd:element name="subscriptionID" type="xsd:string" minOccurs="0"/>
             </xsd:sequence>
          </xsd:extension>
       </xsd:complexContent>
    </xsd:complexType>

    <xsd:element name="QueryTooComplexException" type="epcisq:QueryTooComplexException"/>
    <xsd:complexType name="QueryTooComplexException">
       <xsd:complexContent>
          <xsd:extension base="epcisq:EPCISException">
             <xsd:sequence/>
          </xsd:extension>
       </xsd:complexContent>
    </xsd:complexType>
```

```
  <xsd:element name="SubscriptionControlsException"
type="epcisq:SubscriptionControlsException"/>
  <xsd:complexType name="SubscriptionControlsException">
     <xsd:complexContent>
        <xsd:extension base="epcisq:EPCISException">
           <xsd:sequence/>
        </xsd:extension>
     </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="SubscribeNotPermittedException"
type="epcisq:SubscribeNotPermittedException"/>
  <xsd:complexType name="SubscribeNotPermittedException">
     <xsd:complexContent>
        <xsd:extension base="epcisq:EPCISException">
           <xsd:sequence/>
        </xsd:extension>
     </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="SecurityException" type="epcisq:SecurityException"/>
  <xsd:complexType name="SecurityException">
     <xsd:complexContent>
        <xsd:extension base="epcisq:EPCISException">
           <xsd:sequence/>
        </xsd:extension>
     </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="ValidationException" type="epcisq:ValidationException"/>
  <xsd:complexType name="ValidationException">
     <xsd:complexContent>
        <xsd:extension base="epcisq:EPCISException">
           <xsd:sequence/>
        </xsd:extension>
     </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="ImplementationException"
               type="epcisq:ImplementationException"/>
  <xsd:complexType name="ImplementationException">
     <xsd:complexContent>
        <xsd:extension base="epcisq:EPCISException">
           <xsd:sequence>
              <xsd:element name="severity"
                           type="epcisq:ImplementationExceptionSeverity"/>
              <xsd:element name="queryName" type="xsd:string" minOccurs="0"/>
              <xsd:element name="subscriptionID" type="xsd:string" minOccurs="0"/>
           </xsd:sequence>
         </xsd:extension>
     </xsd:complexContent>
   </xsd:complexType>

   <xsd:simpleType name="ImplementationExceptionSeverity">
     <xsd:restriction base="xsd:NCName">
         <xsd:enumeration value="ERROR"/>
         <xsd:enumeration value="SEVERE"/>
     </xsd:restriction>
   </xsd:simpleType>

</xsd:schema>
```

11.2 SOAP/HTTP Binding for the Query Control Interface

The following is a Web Service Description Language (WSDL) 1.1 [WSDL1.1] specification defining the standard SOAP/HTTP binding of the EPCIS Query Control Interface. An EPCIS implementation MAY provide a SOAP/HTTP binding of the EPCIS

Query Control Interface; if a SOAP/HTTP binding is provided, it SHALL conform to the following WSDL. This SOAP/HTTP binding is compliant with the WS-I Basic Profile Version 1.0 [WSI]. This binding builds upon the schema defined in Section 11.1.

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

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


<!-- EPCIS QUERY SERVICE DEFINITIONS -->
<wsdl:definitions
    targetNamespace="urn:epcglobal:epcis:wsdl:1"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:apachesoap="http://xml.apache.org/xml-soap"
    xmlns:epcis="urn:epcglobal:epcis:xsd:1"
    xmlns:epcisq="urn:epcglobal:epcis-query:xsd:1"
    xmlns:epcglobal="urn:epcglobal:xsd:1"
    xmlns:impl="urn:epcglobal:epcis:wsdl:1"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <wsdl:documentation>
    <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>
    </epcglobal:specification>
  </wsdl:documentation>

  <!-- EPCISSERVICE TYPES -->
  <wsdl:types>
     <xsd:schema targetNamespace="urn:epcglobal:epcis:wsdl:1"
                 xmlns:impl="urn:epcglobal:epcis:wsdl:1"
                 xmlns:xsd="http://www.w3.org/2001/XMLSchema">

       <xsd:import
            namespace="urn:epcglobal:xsd:1"
            schemaLocation="EPCglobal.xsd"/>
       <xsd:import
            namespace="urn:epcglobal:epcis:xsd:1"
            schemaLocation="EPCglobal-epcis-1_0.xsd"/>
       <xsd:import
            namespace="urn:epcglobal:epcis-query:xsd:1"
            schemaLocation="EPCglobal-epcis-query-1_0.xsd"/>
    </xsd:schema>
  </wsdl:types>

  <!-- EPCIS QUERY SERVICE MESSAGES -->

  <wsdl:message name="getQueryNamesRequest">
      <wsdl:part name="parms" element="epcisq:GetQueryNames"/>
  </wsdl:message>
  <wsdl:message name="getQueryNamesResponse">
      <wsdl:part name="getQueryNamesReturn" element="epcisq:GetQueryNamesResult"/>
  </wsdl:message>
```

```
    <wsdl:message name="subscribeRequest">
      <wsdl:part name="parms" element="epcisq:Subscribe"/>
    </wsdl:message>
    <wsdl:message name="subscribeResponse">
        <wsdl:part name="subscribeReturn" element="epcisq:SubscribeResult"/>
    </wsdl:message>

    <wsdl:message name="unsubscribeRequest">
        <wsdl:part name="parms" element="epcisq:Unsubscribe"/>
    </wsdl:message>
    <wsdl:message name="unsubscribeResponse">
        <wsdl:part name="unsubscribeReturn" element="epcisq:UnsubscribeResult"/>
    </wsdl:message>

    <wsdl:message name="getSubscriptionIDsRequest">
        <wsdl:part name="parms" element="epcisq:GetSubscriptionIDs"/>
    </wsdl:message>
    <wsdl:message name="getSubscriptionIDsResponse">
        <wsdl:part name="getSubscriptionIDsReturn"
element="epcisq:GetSubscriptionIDsResult"/>
    </wsdl:message>

    <wsdl:message name="pollRequest">
        <wsdl:part name="parms" element="epcisq:Poll"/>
    </wsdl:message>
    <wsdl:message name="pollResponse">
        <wsdl:part name="pollReturn" element="epcisq:QueryResults"/>
    </wsdl:message>

    <wsdl:message name="getStandardVersionRequest">
        <wsdl:part name="parms" element="epcisq:GetStandardVersion"/>
    </wsdl:message>
    <wsdl:message name="getStandardVersionResponse">
        <wsdl:part name="getStandardVersionReturn"
element="epcisq:GetStandardVersionResult"/>
    </wsdl:message>

    <wsdl:message name="getVendorVersionRequest">
        <wsdl:part name="parms" element="epcisq:GetVendorVersion"/>
    </wsdl:message>
    <wsdl:message name="getVendorVersionResponse">
        <wsdl:part name="getVendorVersionReturn" element="epcisq:GetVendorVersionResult"/>
    </wsdl:message>

    <!-- EPCISSERVICE FAULT EXCEPTIONS -->
    <wsdl:message name="DuplicateNameExceptionResponse">
        <wsdl:part name="fault" element="epcisq:DuplicateNameException"/>
    </wsdl:message>
        <!-- QueryValidationException not implemented in EPCIS 1.0
    <wsdl:message name="QueryValidationExceptionResponse">
        <wsdl:part name="fault" element="epcisq:QueryValidationException"/>
    </wsdl:message>
        -->
    <wsdl:message name="InvalidURIExceptionResponse">
        <wsdl:part name="fault" element="epcisq:InvalidURIException"/>
    </wsdl:message>
    <wsdl:message name="NoSuchNameExceptionResponse">
        <wsdl:part name="fault" element="epcisq:NoSuchNameException"/>
    </wsdl:message>
    <wsdl:message name="NoSuchSubscriptionExceptionResponse">
        <wsdl:part name="fault" element="epcisq:NoSuchSubscriptionException"/>
    </wsdl:message>
    <wsdl:message name="DuplicateSubscriptionExceptionResponse">
        <wsdl:part name="fault" element="epcisq:DuplicateSubscriptionException"/>
    </wsdl:message>
    <wsdl:message name="QueryParameterExceptionResponse">
        <wsdl:part name="fault" element="epcisq:QueryParameterException"/>
    </wsdl:message>
    <wsdl:message name="QueryTooLargeExceptionResponse">
```

```
        <wsdl:part name="fault" element="epcisq:QueryTooLargeException"/>
    </wsdl:message>
    <wsdl:message name="QueryTooComplexExceptionResponse">
        <wsdl:part name="fault" element="epcisq:QueryTooComplexException"/>
    </wsdl:message>
    <wsdl:message name="SubscriptionControlsExceptionResponse">
        <wsdl:part name="fault" element="epcisq:SubscriptionControlsException"/>
    </wsdl:message>
    <wsdl:message name="SubscribeNotPermittedExceptionResponse">
        <wsdl:part name="fault" element="epcisq:SubscribeNotPermittedException"/>
    </wsdl:message>
    <wsdl:message name="SecurityExceptionResponse">
        <wsdl:part name="fault" element="epcisq:SecurityException"/>
    </wsdl:message>
    <wsdl:message name="ValidationExceptionResponse">
        <wsdl:part name="fault" element="epcisq:ValidationException"/>
    </wsdl:message>
    <wsdl:message name="ImplementationExceptionResponse">
        <wsdl:part name="fault" element="epcisq:ImplementationException"/>
    </wsdl:message>

    <!-- EPCISSERVICE PORTTYPE -->
    <wsdl:portType name="EPCISServicePortType">

      <wsdl:operation name="getQueryNames">
        <wsdl:input message="impl:getQueryNamesRequest" name="getQueryNamesRequest"/>
        <wsdl:output message="impl:getQueryNamesResponse" name="getQueryNamesResponse"/>
        <wsdl:fault message="impl:SecurityExceptionResponse"
name="SecurityExceptionFault"/>
        <wsdl:fault message="impl:ValidationExceptionResponse"
name="ValidationExceptionFault"/>
        <wsdl:fault message="impl:ImplementationExceptionResponse"
name="ImplementationExceptionFault"/>
    </wsdl:operation>

    <wsdl:operation name="subscribe">
      <wsdl:input message="impl:subscribeRequest" name="subscribeRequest"/>
      <wsdl:output message="impl:subscribeResponse" name="subscribeResponse"/>
      <wsdl:fault message="impl:NoSuchNameExceptionResponse"
name="NoSuchNameExceptionFault"/>
      <wsdl:fault message="impl:InvalidURIExceptionResponse"
name="InvalidURIExceptionFault"/>
      <wsdl:fault message="impl:DuplicateSubscriptionExceptionResponse"
name="DuplicateSubscriptionExceptionFault"/>
      <wsdl:fault message="impl:QueryParameterExceptionResponse"
name="QueryParameterExceptionFault"/>
      <wsdl:fault message="impl:QueryTooComplexExceptionResponse"
name="QueryTooComplexExceptionFault"/>
      <wsdl:fault message="impl:SubscriptionControlsExceptionResponse"
name="SubscriptionControlsExceptionFault"/>
      <wsdl:fault message="impl:SubscribeNotPermittedExceptionResponse"
name="SubscribeNotPermittedExceptionFault"/>
      <wsdl:fault message="impl:SecurityExceptionResponse"
name="SecurityExceptionFault"/>
      <wsdl:fault message="impl:ValidationExceptionResponse"
name="ValidationExceptionFault"/>
      <wsdl:fault message="impl:ImplementationExceptionResponse"
name="ImplementationExceptionFault"/>
    </wsdl:operation>

    <wsdl:operation name="unsubscribe">
      <wsdl:input message="impl:unsubscribeRequest" name="unsubscribeRequest"/>
      <wsdl:output message="impl:unsubscribeResponse" name="unsubscribeResponse"/>
      <wsdl:fault message="impl:NoSuchSubscriptionExceptionResponse"
name="NoSuchSubscriptionExceptionFault"/>
      <wsdl:fault message="impl:SecurityExceptionResponse"
name="SecurityExceptionFault"/>
      <wsdl:fault message="impl:ValidationExceptionResponse"
name="ValidationExceptionFault"/>
```

```
3570        <wsdl:fault message="impl:ImplementationExceptionResponse"
3571    name="ImplementationExceptionFault"/>
3572      </wsdl:operation>
3573
3574      <wsdl:operation name="getSubscriptionIDs">
3575        <wsdl:input message="impl:getSubscriptionIDsRequest"
3576    name="getSubscriptionIDsRequest"/>
3577        <wsdl:output message="impl:getSubscriptionIDsResponse"
3578    name="getSubscriptionIDsResponse"/>
3579        <wsdl:fault message="impl:NoSuchNameExceptionResponse"
3580    name="NoSuchNameExceptionFault"/>
3581        <wsdl:fault message="impl:SecurityExceptionResponse"
3582    name="SecurityExceptionFault"/>
3583        <wsdl:fault message="impl:ValidationExceptionResponse"
3584    name="ValidationExceptionFault"/>
3585        <wsdl:fault message="impl:ImplementationExceptionResponse"
3586    name="ImplementationExceptionFault"/>
3587      </wsdl:operation>
3588
3589      <wsdl:operation name="poll">
3590        <wsdl:input message="impl:pollRequest" name="pollRequest"/>
3591        <wsdl:output message="impl:pollResponse" name="pollResponse"/>
3592        <wsdl:fault message="impl:QueryParameterExceptionResponse"
3593    name="QueryParameterExceptionFault"/>
3594        <wsdl:fault message="impl:QueryTooLargeExceptionResponse"
3595    name="QueryTooLargeExceptionFault"/>
3596        <wsdl:fault message="impl:QueryTooComplexExceptionResponse"
3597    name="QueryTooComplexExceptionFault"/>
3598        <wsdl:fault message="impl:NoSuchNameExceptionResponse"
3599    name="NoSuchNameExceptionFault"/>
3600        <wsdl:fault message="impl:SecurityExceptionResponse"
3601    name="SecurityExceptionFault"/>
3602        <wsdl:fault message="impl:ValidationExceptionResponse"
3603    name="ValidationExceptionFault"/>
3604        <wsdl:fault message="impl:ImplementationExceptionResponse"
3605    name="ImplementationExceptionFault"/>
3606      </wsdl:operation>
3607
3608      <wsdl:operation name="getStandardVersion">
3609        <wsdl:input message="impl:getStandardVersionRequest"
3610    name="getStandardVersionRequest"/>
3611        <wsdl:output message="impl:getStandardVersionResponse"
3612    name="getStandardVersionResponse"/>
3613        <wsdl:fault message="impl:SecurityExceptionResponse"
3614    name="SecurityExceptionFault"/>
3615        <wsdl:fault message="impl:ValidationExceptionResponse"
3616    name="ValidationExceptionFault"/>
3617        <wsdl:fault message="impl:ImplementationExceptionResponse"
3618    name="ImplementationExceptionFault"/>
3619      </wsdl:operation>
3620
3621      <wsdl:operation name="getVendorVersion">
3622        <wsdl:input message="impl:getVendorVersionRequest" name="getVendorVersionRequest"/>
3623        <wsdl:output message="impl:getVendorVersionResponse"
3624    name="getVendorVersionResponse"/>
3625        <wsdl:fault message="impl:SecurityExceptionResponse"
3626    name="SecurityExceptionFault"/>
3627        <wsdl:fault message="impl:ValidationExceptionResponse"
3628    name="ValidationExceptionFault"/>
3629        <wsdl:fault message="impl:ImplementationExceptionResponse"
3630    name="ImplementationExceptionFault"/>
3631      </wsdl:operation>
3632    </wsdl:portType>
3633
3634    <!-- EPCISSERVICE BINDING -->
3635    <wsdl:binding name="EPCISServiceBinding" type="impl:EPCISServicePortType">
3636      <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
3637
3638      <wsdl:operation name="getQueryNames">
3639          <wsdlsoap:operation soapAction=""/>
```

```
            <wsdl:input name="getQueryNamesRequest">
                <wsdlsoap:body
                    use="literal"/>
            </wsdl:input>
            <wsdl:output name="getQueryNamesResponse">
                <wsdlsoap:body
                    use="literal"/>
            </wsdl:output>
            <wsdl:fault name="SecurityExceptionFault">
                <wsdlsoap:fault
                    name="SecurityExceptionFault"
                    use="literal"/>
            </wsdl:fault>
            <wsdl:fault name="ValidationExceptionFault">
                <wsdlsoap:fault
                    name="ValidationExceptionFault"
                    use="literal"/>
            </wsdl:fault>
            <wsdl:fault name="ImplementationExceptionFault">
                <wsdlsoap:fault
                    name="ImplementationExceptionFault"
                    use="literal"/>
            </wsdl:fault>
        </wsdl:operation>

        <wsdl:operation name="subscribe">
            <wsdlsoap:operation soapAction=""/>
            <wsdl:input name="subscribeRequest">
                <wsdlsoap:body
                    use="literal"/>
            </wsdl:input>
            <wsdl:output name="subscribeResponse">
                <wsdlsoap:body
                    use="literal"/>
            </wsdl:output>
            <wsdl:fault name="NoSuchNameExceptionFault">
                <wsdlsoap:fault
                    name="NoSuchNameExceptionFault"
                    use="literal"/>
            </wsdl:fault>
            <wsdl:fault name="InvalidURIExceptionFault">
                <wsdlsoap:fault
                    name="InvalidURIExceptionFault"
                    use="literal"/>
            </wsdl:fault>
            <wsdl:fault name="DuplicateSubscriptionExceptionFault">
                <wsdlsoap:fault
                    name="DuplicateSubscriptionExceptionFault"
                    use="literal"/>
            </wsdl:fault>
            <wsdl:fault name="QueryParameterExceptionFault">
                <wsdlsoap:fault
                    name="QueryParameterExceptionFault"
                    use="literal"/>
            </wsdl:fault>
            <wsdl:fault name="QueryTooComplexExceptionFault">
                <wsdlsoap:fault
                    name="QueryTooComplexExceptionFault"
                    use="literal"/>
            </wsdl:fault>
            <wsdl:fault name="SubscribeNotPermittedExceptionFault">
                <wsdlsoap:fault
                    name="SubscribeNotPermittedExceptionFault"
                    use="literal"/>
            </wsdl:fault>
            <wsdl:fault name="SubscriptionControlsExceptionFault">
                <wsdlsoap:fault
                    name="SubscriptionControlsExceptionFault"
                    use="literal"/>
            </wsdl:fault>
```

```
            <wsdl:fault name="SecurityExceptionFault">
                <wsdlsoap:fault
                    name="SecurityExceptionFault"
                    use="literal"/>
            </wsdl:fault>
            <wsdl:fault name="ValidationExceptionFault">
                <wsdlsoap:fault
                    name="ValidationExceptionFault"
                    use="literal"/>
            </wsdl:fault>
            <wsdl:fault name="ImplementationExceptionFault">
                <wsdlsoap:fault
                    name="ImplementationExceptionFault"
                    use="literal"/>
            </wsdl:fault>
        </wsdl:operation>

        <wsdl:operation name="unsubscribe">
            <wsdlsoap:operation soapAction=""/>
            <wsdl:input name="unsubscribeRequest">
                <wsdlsoap:body
                    use="literal"/>
            </wsdl:input>
            <wsdl:output name="unsubscribeResponse">
                <wsdlsoap:body
                    use="literal"/>
            </wsdl:output>
            <wsdl:fault name="NoSuchSubscriptionExceptionFault">
                <wsdlsoap:fault
                    name="NoSuchSubscriptionExceptionFault"
                    use="literal"/>
            </wsdl:fault>
            <wsdl:fault name="SecurityExceptionFault">
                <wsdlsoap:fault
                    name="SecurityExceptionFault"
                    use="literal"/>
            </wsdl:fault>
            <wsdl:fault name="ValidationExceptionFault">
                <wsdlsoap:fault
                    name="ValidationExceptionFault"
                    use="literal"/>
            </wsdl:fault>
            <wsdl:fault name="ImplementationExceptionFault">
                <wsdlsoap:fault
                    name="ImplementationExceptionFault"
                    use="literal"/>
            </wsdl:fault>
        </wsdl:operation>

        <wsdl:operation name="getSubscriptionIDs">
            <wsdlsoap:operation soapAction=""/>
            <wsdl:input name="getSubscriptionIDsRequest">
                <wsdlsoap:body
                    use="literal"/>
            </wsdl:input>
            <wsdl:output name="getSubscriptionIDsResponse">
                <wsdlsoap:body
                    use="literal"/>
            </wsdl:output>
            <wsdl:fault name="NoSuchNameExceptionFault">
                <wsdlsoap:fault
                    name="NoSuchNameExceptionFault"
                    use="literal"/>
            </wsdl:fault>
            <wsdl:fault name="SecurityExceptionFault">
                <wsdlsoap:fault
                    name="SecurityExceptionFault"
                    use="literal"/>
            </wsdl:fault>
            <wsdl:fault name="ValidationExceptionFault">
```

```
            <wsdlsoap:fault
                name="ValidationExceptionFault"
                use="literal"/>
        </wsdl:fault>
        <wsdl:fault name="ImplementationExceptionFault">
            <wsdlsoap:fault
                name="ImplementationExceptionFault"
                use="literal"/>
        </wsdl:fault>
    </wsdl:operation>

    <wsdl:operation name="poll">
        <wsdlsoap:operation soapAction=""/>
        <wsdl:input name="pollRequest">
            <wsdlsoap:body
                use="literal"/>
        </wsdl:input>
        <wsdl:output name="pollResponse">
            <wsdlsoap:body
                use="literal"/>
        </wsdl:output>
        <wsdl:fault name="QueryParameterExceptionFault">
            <wsdlsoap:fault
                name="QueryParameterExceptionFault"
                use="literal"/>
        </wsdl:fault>
        <wsdl:fault name="QueryTooComplexExceptionFault">
            <wsdlsoap:fault
                name="QueryTooComplexExceptionFault"
                use="literal"/>
        </wsdl:fault>
        <wsdl:fault name="QueryTooLargeExceptionFault">
            <wsdlsoap:fault
                name="QueryTooLargeExceptionFault"
                use="literal"/>
        </wsdl:fault>
        <wsdl:fault name="NoSuchNameExceptionFault">
            <wsdlsoap:fault
                name="NoSuchNameExceptionFault"
                use="literal"/>
        </wsdl:fault>
        <wsdl:fault name="SecurityExceptionFault">
            <wsdlsoap:fault
                name="SecurityExceptionFault"
                use="literal"/>
        </wsdl:fault>
        <wsdl:fault name="ValidationExceptionFault">
            <wsdlsoap:fault
                name="ValidationExceptionFault"
                use="literal"/>
        </wsdl:fault>
        <wsdl:fault name="ImplementationExceptionFault">
            <wsdlsoap:fault
                name="ImplementationExceptionFault"
                use="literal"/>
        </wsdl:fault>
    </wsdl:operation>

    <wsdl:operation name="getStandardVersion">
        <wsdlsoap:operation soapAction=""/>
        <wsdl:input name="getStandardVersionRequest">
            <wsdlsoap:body
                use="literal"/>
        </wsdl:input>
        <wsdl:output name="getStandardVersionResponse">
            <wsdlsoap:body
                use="literal"/>
        </wsdl:output>
        <wsdl:fault name="SecurityExceptionFault">
            <wsdlsoap:fault
```

```
                name="SecurityExceptionFault"
                use="literal"/>
        </wsdl:fault>
        <wsdl:fault name="ValidationExceptionFault">
            <wsdlsoap:fault
                name="ValidationExceptionFault"
                use="literal"/>
        </wsdl:fault>
        <wsdl:fault name="ImplementationExceptionFault">
            <wsdlsoap:fault
                name="ImplementationExceptionFault"
                use="literal"/>
        </wsdl:fault>
    </wsdl:operation>

    <wsdl:operation name="getVendorVersion">
        <wsdlsoap:operation soapAction=""/>
        <wsdl:input name="getVendorVersionRequest">
            <wsdlsoap:body
                use="literal"/>
        </wsdl:input>
        <wsdl:output name="getVendorVersionResponse">
            <wsdlsoap:body
                use="literal"/>
        </wsdl:output>
        <wsdl:fault name="SecurityExceptionFault">
            <wsdlsoap:fault
                name="SecurityExceptionFault"
                use="literal"/>
        </wsdl:fault>
        <wsdl:fault name="ValidationExceptionFault">
            <wsdlsoap:fault
                name="ValidationExceptionFault"
                use="literal"/>
        </wsdl:fault>
        <wsdl:fault name="ImplementationExceptionFault">
            <wsdlsoap:fault
                name="ImplementationExceptionFault"
                use="literal"/>
        </wsdl:fault>
    </wsdl:operation>

  </wsdl:binding>

  <!-- EPCISSERVICE -->
  <wsdl:service name="EPCglobalEPCISService">
    <wsdl:port binding="impl:EPCISServiceBinding" name="EPCglobalEPCISServicePort">
    <!-- The address shown below is an example; an implementation MAY specify
         any port it wishes
     -->
      <wsdlsoap:address
         location="http://localhost:6060/axis/services/EPCglobalEPCISService"/>
    </wsdl:port>
  </wsdl:service>

</wsdl:definitions>
```

11.3 AS2 Binding for the Query Control Interface

This section defines a binding of the EPCIS Query Control Interface to AS2 [RFC4130]. An EPCIS implementation MAY provide an AS2 binding of the EPCIS Query Control Interface; if an AS2 binding is provided it SHALL conform to the provisions of this section. For the purposes of this binding, a “query client” is an EPCIS Accessing Application that wishes to issue EPCIS query operations as defined in Section 8.2.5, and

a "query server" is an EPCIS Repository or other system that carries out such operations on behalf of the query client.

A query server SHALL provide an HTTP URL through which it receives messages from a query client in accordance with [RFC4130]. A message sent by a query client to a query server SHALL be an XML document whose root element conforms to the `EPCISQueryDocument` element as defined by the schema in Section 11.1. The element immediately nested within the `EPCISBody` element SHALL be one of the elements corresponding to a EPCIS Query Control Interface method request (i.e., one of `Subscribe`, `Unsubscribe`, `Poll`, etc.). The permitted elements are listed in the table below. If the message sent by the query client fails to conform to the above requirements, the query server SHALL respond with a `ValidationException` (that is, return an `EPCISQueryDocument` instance where the element immediately nested within the `EPCISBody` is a `ValidationException`).

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

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

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

A query server SHALL respond to each message sent by a query client by delivering a response message to the URL provided by the query client, in accordance with [RFC4130]. A response message sent by a query server SHALL be an XML document whose root element conforms to the `EPCISQueryDocument` element as defined by the schema in Section 11.1. The element immediately nested within the `EPCISBody` element SHALL be one of the elements shown in the following table, according to the element that was provided in the corresponding request:

Request Element	Permitted Return Elements

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

Request Element	Permitted Return Elements
Subscribe	SubscribeResult NoSuchNameException InvalidURIException DuplicateSubscriptionException QueryParameterException QueryTooComplexException SubscriptionControlsException SubscribeNotPermittedException SecurityException ValidationException ImplementationException
Unsubscribe	UnsubscribeResult NoSuchSubscriptionException SecurityException ValidationException ImplementationException
GetSubscriptionIDs	GetSubscriptionIDsResult NoSuchNameException SecurityException ValidationException ImplementationException
Poll	QueryResults QueryParameterException QueryTooLargeException QueryTooComplexException NoSuchNameException SecurityException ValidationException ImplementationException
GetStandardVersion	GetStandardVersionResult SecurityException ValidationException ImplementationException
GetVendorVersion	GetVendorVersionResult SecurityException ValidationException ImplementationException

3949

3950 The query server SHALL include the Standard Business Document Header within the
3951 EPCISHeader element. The query server SHALL include within the Standard Business
3952 Document Header the `BusinessScope` element containing a `Scope` element
3953 containing a `CorrelationInformation` element containing a

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

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

Digital Certificate Requirements

Requirement 1

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

Requirement 2

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

Requirement 3

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

Requirement 4

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

Requirement 5

The Signature Hash algorithm used SHALL be SHA1.

Configuration Requirement

Requirement 6

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

Recommendations

Recommendation 1 – MDN Request Option

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

Recommendation 2 – MDN Delivery

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

Recommendation 3 – Delivery Retry with Asynchronous MDNs Requested

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

Recommendation 4 – Delivery Retry for AS2

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

Recommendation 5 – Message Resubmission

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

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

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

Recommendation 7 – AS2 Header

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

Recommendation 8 - SMTP

[not applicable]

Recommendation 9 - Compression

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

Recommendation 10 – Digital Certificate Characteristics

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

Recommendation 11 – Common Digital Certificate for Encryption & Signature

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

Recommendation 12 – Digital Certificate Validity Period

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

Recommendation 13 – Digital Certificate – Automated Exchange

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

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

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

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

Recommendation 15 – Duplicate AS2 Messages

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

Recommendation 15 – Technical Support

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

11.4 Bindings for Query Callback Interface

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

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

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

11.4.1 General Considerations for all XML-based Bindings

The following applies to all XML-based bindings of the Query Callback Interface, including the bindings specified in Sections 11.4.2, 11.4.3, and 11.4.4.

The payload delivered to the recipient SHALL be an XML document conforming to the schema specified in Section 11.1. Specifically, the payload SHALL be an `EPCISQueryDocument` instance whose `EPCISBody` element contains one of the three elements shown in the table below, according to the method of the Query Callback Interface being invoked:

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

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

11.4.2 HTTP Binding of the Query Callback Interface

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

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

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

where

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

The EPCIS implementation SHALL deliver query results by sending an HTTP POST request to receiver designated in the URI, where *`remainder-of-URL`* is included in the HTTP `request-line` (as defined in [RFC2616]), and where the payload is an XML document as specified in Section 11.4.1.

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

11.4.3 HTTPS Binding of the Query Callback Interface

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

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

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

where

- *`host`* is the DNS name or IP address of the host where the receiver is listening for incoming HTTP connections.
- *`port`* is the TCP port on which the receiver is listening for incoming HTTP connections. The port and the preceding colon character may be omitted, in which case the port defaults to 443.

- *remainder-of-URL* is the URL to which an HTTP POST operation will be directed.

The EPCIS implementation SHALL deliver query results by sending an HTTP POST request to receiver designated in the URI, where *remainder-of-URL* is included in the HTTP `request-line` (as defined in [RFC2616]), and where the payload is an XML document as specified in Section 11.4.1.

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

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

11.4.4 AS2 Binding of the Query Callback Interface

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

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

`as2:`*`remainder-of-URI`*

where

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

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

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

The EPCIS implementation SHALL deliver query results by sending an AS2 message in accodance with [RFC4130]. The AS2 message payload SHALL be an XML document as specified in Section 11.4.1.

Both the EPCIS Service and the receipient of standing query results SHALL comply with the Requirements and SHOULD comply with the Recommendations listed in the GS1 document "EDIINT AS1 and AS2 Transport Communications Guidelines" [EDICG] For reference, the relevant portions of this document are reproduced in Section 11.3.

12 References

Normative references:

[ALE1.0] EPCglobal, "The Application Level Events (ALE) Specification, Version 1.0," EPCglobal Standard Specification, September 2005, http://www.epcglobalinc.org/standards_technology/EPCglobal_ApplicationALE_Specification_v112-2005.pdf.

[EDICG] GS1, "EDIINT AS1 and AS2 Transport Communications Guidelines," GS1 Technical Document, February 2006, http://www.ean-ucc.org/global_smp/documents/zip/EDIINT%20AS2/EDIINT_AS1-AS2_Transport_Comm_Guidelines_2006.pdf.

[ISODir2] ISO, "Rules for the structure and drafting of International Standards (ISO/IEC Directives, Part 2, 2001, 4th edition)," July 2002.

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

[RFC2141] R. Moats, "URN Syntax," Internet Engineering Task Force Request for Comments RFC-2141, May 1997, http://www.ietf.org/rfc/rfc2141.txt.

[RFC2246] T. Dierks, C. Allen, "The TLS Protocol, Version 1.0," RFC2246, January 1999, http://www.ietf.org/rfc/rfc2246.

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

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

[RFC2818] E. Escorla, "HTTP Over TLS," RFC2818, May 2000, http://www.ietf.org/rfc/rfc2818.

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

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

[SBDH] United Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT), “Standard Business Document Header Technical Specification, Version 1.3,” June 2004, http://www.disa.org/cefact-groups/atg/downloads/.

[TDS1.3] EPCglobal, “EPCglobal Tag Data Standards Version 1.3,” EPCglobal Standard Specification, March 2006, http://www.epcglobalinc.org/standards_technology/Ratified%20Spec%20March%208%202006.pdf.

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

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

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

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

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

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

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

Non-normative references:

[EPCAF] K. R. Traub et al, “EPCglobal Architecture Framework,” EPCglobal technical document, July 2005, http://www.epcglobalinc.org/standards_technology/Final-epcglobal-arch-20050701.pdf.

[EPCIS-User] K. Traub, S. Rehling, R. Swan, G. Gilbert, J. Chiang, J. Navas, M. Mealling, S. Ramachandran, “EPC Information Services (EPCIS) User Definition,” EPCglobal Working Draft, October 2004.

13 Acknowledgement of Contributors and Companies Opt'd-in during the Creation of this Standard (Informative)

Disclaimer

Whilst every effort has been made to ensure that this document and the information contained herein are correct, EPCglobal and any other party involved in the creation of the document hereby state that the document is provided on an "as is" basis without warranty, either expressed or implied, including but not limited to any warranty that the use of the information herein with not infringe any rights, of accuracy or fitness for purpose, and hereby disclaim any liability, direct or indirect, for damages or loss relating to the use of the document.

Below is a list of more active participants and contributors in the development of EPCIS v1.0. This list does not acknowledge those who only monitored the process or those who chose not to have their name listed here. The participants listed below generated emails, attended face-to-face meetings and conference calls that were associated with the development of this Standard.

First Name	Last Name	Company	
Craig	Asher	IBM	Co-Chair
Greg	Gibert	Verisign	Co-Chair
Richard	Swan	T3Ci	Co-Chair
Ken	Traub	BEA Systems; ConnecTerra	Specification Editor
Gena	Morgan	EPCglobal, Inc.	WorkGroup Facilitator
Chi-Hyeong	Ahn	Ceyon Technology Co., Ltd	
Umair	Akeel	IBM	
John	Anderla	Kimberly-Clark Corp	
Richard	Bach	Globe Ranger	

First Name	Last Name	Company	
Scott	Barvick	Reva Systems	
Sylvanus	Bent	Bent Systems, Inc.	
Hersh	Bhargava	Rafcor	
Chet	Birger	ConnecTerra	
Bud	Biswas	Polaris Networks	
Prabhudda	Biswas	Oracle Corporation	
Havard	Bjastad	Tracetracker	
Joe	Bohning	Nestle Purina	
Al	Bottner	UNITED PARCEL SERVICE (UPS)	
Joe	Bradley	Sun Microsystems	
Leo	Burstein	Gillette; Procter & Gamble	
Anit	Chakraborty	Oracle Corporation	
Chia	Chang	Sun Microsystems	
Ying-Hung	Chang	Acer Cybercenter Service Inc.	
Martin	Chen	SAP	
Nagesh	Chigurupati	VeriSign	
Christian	Clauss	IBM	
John	Cooper	Kimberly-Clark Corp	
Valir-Alin	Crisan	IBM	
Mustafa	Dohadwala	Shipcom Wireless, Inc.	
John	Duker	Procter & Gamble	
Igor	Elbert	Sensitech	
Ronny	Fehling	Oracle Corporation	
Akira	Fujinami	Internet Initiative Japan, Inc.	
Tony	Gallo	Real Time Systems	
Manish	Gambhir		
Cesar	Gemayel	Sensitech	
Eric	Gieseke	BEA Systems	
Greg	Gilbert	Manhattan Associates	
Graham	Gillen	Verisign	
John	Gravitis	Allumis	
Yuichiro	Hanawa	Mitsui	
Mark	Harrison	Auto-ID Labs - Cambridge	
Jeremy	Helm	ACSIS	

First Name	Last Name	Company	
Barba	Hickman	Intermec	
Manju	James	BEA Systems	
Paul	Jatkowski		
Jennifer	Kahn	IBM	
Howard	Kapustein	Manhattan Associates	
Sean	Lockhead	GS1 US	
Paul	Lovvik	Sun Microsystems	
Midori	Lowe	Nippon Telegraph & Telephone Corp (NTT)	
Dave	Marzouck	SAP	
Andrew	McGrath	Manhattan Associates	
MIchael	Mealling	Verisign; Refactored Networks	
Stephen	Miles	Auto-ID Labs - MIT	
Tim	Milne	Target	
Dale	Moberg	AXWAY/formerly Cyclone	
Stephen	Morris	Printronix	
Ron	Moser	Wal-Mart	
Don	Mowery	Nestle	
Doug	Naal	Altria Group, Inc./Kraft Foods	
David	Nesbitt	Vue Technology	
Shigeki	Ohtsu	Internet Initiative Japan, Inc.	
Ted	Osinski	MET Labs	
Jong	Park	Tibco	
Ju-Hyun	Park	Samsung SDS	
Sung Gong	Park	Metarights	
Eliot	Polk	Reva Systems	
Mike	Profit	Verisign	
Sridhar	Ramachandran	OAT Systems	
Ajay	Ramachandron		
Karen	Randall	Johnson & Johnson	
Steve	Rehling	Procter & Gamble	
Nagendra	Revanur	T3Ci Incorporated	
Thomas	Rumbach	SAP	
Uday	Sadhukhan	Polaris Networks	

First Name	Last Name	Company	
Hares	Sangani	Hubspan, Inc.	
Puneet	Sawhney	CHEP	
Rick	Schendel	Target	
Chris	Shabsin	BEA Systems	
Bhavesh	Shah	Abbott Laboratories	
Harshal	Shah	Oracle Corporation	
Dong Cheul	Shin	Metarights	
Sung-hak	Song	Samsung SDS	
Ashley	Stephenson	Reva Systems	
Nikola	Stojanovic	GS1 US	
Jim	Sykes	Savi Technology	
Hiroki	Tagato	NEC Corporation	
Diane	Taillard	GS1 France	
Neil	Tan	UPS	
Zach	Thom	Unilever	
Frank	Thompson	Afilias Canada Corp	
Frank	Tittel	Gedas Deutschland GmbH	
Bryan	Tracey	Globe Ranger	
Hsi-Lin	Tsai	Acer Cybercenter Service Inc.	
Richard	Ulrich	Walmart	
David	Unge		
Steve	Vazzano	1Sync	
Vasanth	Velusamy	Supply Insight, Inc.	
Dan	Wallace		
Jie	Wang	True Demand Software (fka-Truth Software)	
John	Williams	Auto-ID Labs - MIT	
Michael	Williams	Hewlett-Packard Co. (HP)	
Steve	Winkler	SAP	
Katsuyuki	Yamashita	Nippon Telegraph & Telephone Corp (NTT)	
Patrick	Yee	Hubspan, Inc.	
Angela	Zilmer	Kimberly-Clark Corp	

4264

The following list in corporate alphabetical order contains all companies that were opt'd-in to the EPCIS Phase 2 Working Group and have signed the EPCglobal IP Policy.

Company
1Sync
7iD (formerly EOSS GmbH)
Abbott Laboratories
Accenture
Acer Cybercenter Service Inc.
ACSIS
Adtio Group Limited
Afilias Canada Corp
Allixon
Allumis
Altria Group, Inc./Kraft Foods
Alvin Systems
AMCO TEC International Inc.
Applied Wireless (AWID)
Ark Tech Ltd
Auto-ID Labs - ADE
Auto-ID Labs - Cambridge
Auto-ID Labs - Fudan University
Auto-ID Labs - ICU
Auto-ID Labs - Japan
Auto-ID Labs - MIT
Auto-ID Labs - Univerisity of St Gallen
Avicon
AXWAY/formerly Cyclone
BEA Systems
Beijing Futianda Technology Co. Ltd.
Benedicta
Bent Systems, Inc.
Best Buy
Bristol Myers Squibb
British Telecom
Cactus Commerce
Campbell Soup Company
Cap Gemini Ernst & Young
Cardinal Health
Ceyon Technology Co., Ltd
CHEP
Cisco
City Univ of Hong Kong
Code Plus, Inc.
Cognizant Technology Solutions
Collaborative Exchange/Techno Solutions

Company
Commercial Development Fund
Computer Network Info Cntr.
Convergence Sys Ltd
Dai Nippon Printing
DEERE & COMPANY (John Deere)
Denso Wave Inc
Dongguk University
ecash corporation
ECO, Inc.
Electronics and Telecommunication Research Institute (ETRI)
EPCglobal, Inc.
EPCglobal US
Frameworx, Inc.
France Telecom
Fujitsu Ltd
Gedas Deutschland GmbH
Glaxo Smith Kline
Globe Ranger
Goliath Solutions
GS1 Australia EAN
GS1 Brazil
GS1 China
GS1 China
GS1 Colombia
GS1 France
GS1 Germany (CCG)
GS1 Hong Kong
GS1 Japan
GS1 Netherlands (EAN.nl)
GS1 Poland Inst of Lgstcs & Wrhsng
GS1 Singapore (Singapore Council)
GS1 South Korea
GS1 Sweden AB (EAN)
GS1 Switzerland
GS1 Taiwan (EAN)
GS1 Thailand (EAN)
GS1 UK
GS1 US
Hewlett-Packard Co. (HP)
Hubspan, Inc.
IBM
Icare Research Institute
iControl, Inc.
Impinj
Indicus Software Pvt Ltd
Indyon GmbH

Company
Infratab
Institute for Information Industry
Insync Software, Inc.
Intelleflex
Intermec
Internet Initiative Japan, Inc.
Johnson & Johnson
Kimberly-Clark Corp
KL-NET
Korea Computer Servs, Ltd
KTNET - KOREA TRADE NETWORK
LIT (Research Ctr for Logistics Info Tech)
Loftware, Inc.
Manhattan Associates
McKesson
MET Labs
Metarights
Metro
Microelectronics Technology, Inc.
Mindsheet Ltd
Mitsui
Mstar Semiconductor
MUL Services
NCR
NEC Corporation
Nestle
Nestle Purina
Nippon Telegraph & Telephone Corp (NTT)
NOL Group (APL Ltd.) (Neptune Orient Lines)
Nomura Research Institute
NORSK Lastbaerer Pool AS
NORTURA BA
NXP Semiconductors
Omnitrol Networks, Inc.
Oracle Corporation
Panda Logistics Co.Ltd
Pango Networks, Inc.
Patni Computer Systems
PepsiCo
Polaris Networks
Pretide Technology, Inc.
Printronix
Procter & Gamble
Provectus Technologia Ind Com Ltd
Psion Teklogix Inc.
Q.E.D. Systems
Rafcore Systems Inc.

Company
RetailTech
Reva Systems
RFIT Solutions GmbH
RFXCEL Corp
Rush Tracking Systems
Samsung Electronics
Sanion Co Ltd
SAP
Savi Technology
Schering-Plough
Schneider National
Sedna Systems, Ltd.
Sensitech
Shipcom Wireless, Inc.
Skandsoft Technologies Pvt.Ltd.
SMART LABEL SOLUTIONS, LLC.
Sterling Commerce
Sun Microsystems
Supply Insight, Inc.
SupplyScape
T3C Incorporated
Target
Tesco
The Boeing Company
ThingMagic, LLC
Tibco
Toppan Printing Co
Toray International, Inc.
Tracetracker
True Demand Software (fka-Truth Software)
TTA Telecommunications Technology Association
Tyco / ADT
Unilever
Unisys
Unitech Electronics Co., Ltd.
UNITED PARCEL SERVICE (UPS)
Ussen Limited Company
VeriSign
Vue Technology
Wal-Mart
Wish Unity (formerly Track-IT RFID)
Yuen Foong Yu Paper

4269

4270