BEA WebLogic RFID Enterprise Server™

Understanding the Event, Master Data, and Data Exchange Services

Version 2.0
Revised: October 12, 2006

Copyright

Restricted Rights Legend

Trademarks and Service Marks

Contents

CHAPTER 1

Introduction and Roadmap

The following sections describe the audience for and organization of this document:

Document Scope and Audience

This document provides an introduction to, and an architectural description of, three services that implement the EPC Information Services (EPCIS):

- Event Service (captures and stores event data)
- Master Data Service (associates business-context information with event data)
- Data Exchange Service (queries event data, can use master data to filter results)

A fourth service, the Reporting Service, is described briefly in this manual but covered fully in *Generating WebLogic RFID Enterprise Server Reports*.

The manual describes the BEA implementation of these services, and BEA extensions to these services.

The intended audience is system administrators who will administer these services and thus need to understand how they work, and for business users who will interact with them.

You can find procedures for configuring these services at installation time in *Installing WebLogic RFID Enterprise Server*. See "Services: Data Flow, GUI, and Name Space Summary" on page 2-5 for information about the relevant administration consoles.

Guide to This Document

This document is organized as follows:

- This chapter, "Introduction and Roadmap," describes the scope of this document and related information.
- "Services Overview," provides an overview figure showing the interaction among the services, and briefly introduces each service.
- "Event Service," describes the types of events that are sent from the edge to the capture interface on the RFID Enterprise Server for storage in the event repository.
- "Master Data Service," describes the relationship between events and master data; defines master data types, attributes, and children; and provides tables listing the default master data types and attributes that are available at installation time.
- "Data Exchange Service," describes the set of query parameters available through the subscription interface.
- "Capture and Query APIs," describes the methods for capturing event data from the edge, and for querying stored event and master data.

Related Documentation

This manual is part of the WebLogic RFID Enterprise Server documentation set, which also includes the following documents and online help:

- *WebLogic RFID Enterprise Server Product Overview* provides an overview of the WebLogic RFID Enterprise Server components and architecture.
- *Installing WebLogic RFID Enterprise Server* describes how to install and configure WebLogic RFID Enterprise Server.
- *Understanding the Event, Master Data, and Data Exchange Services* (this manual) describes the services that implement the EPC Information Services (EPCIS).
- *Query Subscription Administration Console Online Help* describes how use the Query Subscription Administration Console to create and manage subscriptions: queries that run

at designated times and send results to specific destinations. This console is the user interface to the Data Exchange Service.

- *Master Data Administration Console Online Help* describes how to use the Master Data Administration Console to create and work with master data types and master data entries.This console is the user interface to the Master Data Service.
- *Generating WebLogic RFID Enterprise Server Reports* describes how to use the RFID Enterprise Server Reporting Service to display predefined RFID reports in a Web browser. This console is the user interface to the Reporting Service.
- *Edge Server Administration Console Online Help* describes how to use the RFID Edge Server Administration Console to view and manage RFID Edge Servers in the enterprise.
- *Using the Serial Number Assignment Service* describes how to use the RFID Enterprise Server Serial Number Assignment service to provide pools of RFID serial numbers for assigning to EPC tags.
- *Using the Telemetry Console Extension* describes how to use the Telemetry Console Extension for graphically presenting real-time Edge Server and RFID device telemetry data. The Telemetry Console Extension is part of the RFID Edge Server Administration Console.
- *Release Notes* lists known problems and workarounds in this release of the *RFID Enterprise Server*.

New and Changed Features

WebLogic RFID Enterprise Server 2.0 extends the EPCIS Query Interface provided by the WebLogic RFID Enterprise Server 1.0 and 1.1. The 2.0 release maintains backwards compatibility with the 1.0 and 1.1 versions.

The important new EPCIS-related features for 2.0 are:

- Support for subscription-based report generation
- Support for query extensions that specify master data name/value pairs
- Support for creation, lookup, and removal of query subscriptions

These features are part of the Data Exchange Service.

CHAPTER 2

Services Overview

The WebLogic RFID Enterprise Server processes incoming RFID tag data, adds business-context information, and provides customized reports to business partners. The Enterprise Server can process data from many sources and turn this raw data into useful information which businesses can use for tracking and decision making purposes.

Applications running on the edge, such as the WebLogic RFID Edge Server, send tag event data to the RFID Enterprise Server. This data provides basic information about when and where an RFID tag was read, and depending on the event type, can contain additional information. The Event Service on the Enterprise Server receives this data from the edge, stores it, and makes it available for queries.

Using the master data service, authorized users can create information called master data: human-readable, business-context information that can be associated with event data. This association makes the information encoded in event data both more understandable to humans and more useful as a business modeling tool. The Master Data Administration Console is the graphical user interface for the master data service.

The data exchange service processes queries and sends the results to internal or external destinations. An authorized user can create a subscription for an internal or external client. Each subscription defines a set of query parameters, the times and dates when the query will be run, and the destination where the results will be sent. The Query Subscription Administration Console is the graphical user interface for the data exchange service.

Note: The Reporting Service, which provides GUI access to event and master data reports, is described in *Generating WebLogic RFID Enterprise Server Reports*.

In summary, the event service captures incoming information from the edge, either from a WebLogic RFID Edge Server or from a custom RFID application; the master data service lets you associate business context information with that event data; and the data exchange service lets you share this information within your company and with your business partners.

Together these services provide the ability to integrate raw RFID data with business context information while tracking tagged items as they move from manufacturer to shipper to warehouse to retail store. In terms of simple event data generated by RFID readers, capturing large amounts of raw data is easy; WebLogic RFID Enterprise Server provides the much more difficult but useful services that distill the captured event data, associate master data (business-context information) with it, and then query both sets of data to provide a human-readable history of locations and transactions for an item or group of items.

Note: BEA's implementation of these services is based on the technically complete, but not formally ratified as of October 2006, EPCglobal *EPC Information Services (EPCIS) Version 1.0 Specification*. A currently available document, *The EPCglobal Architecture Framework*, provides a high-level description of the EPCIS architecture; it is available at:

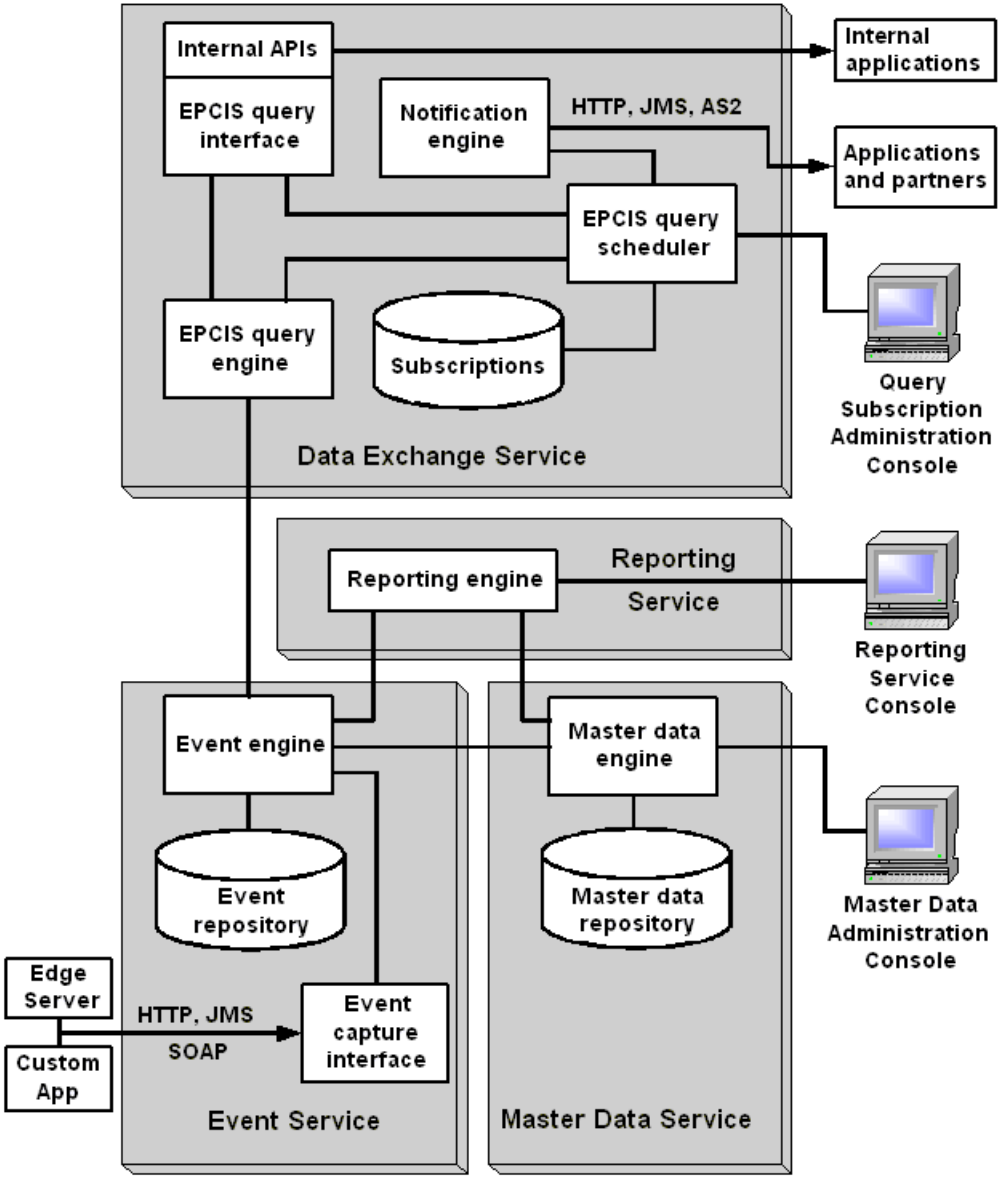
http://www.epcglobalinc.org/standards_technology/specifications.html

Figure 2-1 shows how the services interact within the WebLogic RFID Enterprise Server framework.

The following sections provide a brief introduction to each service:

- "Event Service" on page 2-4
- "Master Data Service" on page 2-4
- "Data Exchange Service" on page 2-5
- "Reporting Service" on page 2-5

Figure 2-1 Event, Master Data, Data Exchange, and Reporting Services



Event Service

The event service captures RFID event data sent from the edge, stores it in the event repository, and makes the data available for retrieval by the data exchange service and the reporting service. Incoming data is formatted as XML; supported transport protocols for incoming data are JMS (queues), HTTP, and SOAP.

The edge is one or more systems such as a WebLogic RFID Edge Server or a custom RFID application that can process information from RFID readers and export event data using the defined event types:

- An *ObjectEvent* captures information about an event pertaining to one or more physical objects identified by EPCs.
- An *AggregationEvent* describes objects that are physically related to one another such that there is a *containing (parent) entity* (for example, a pallet) and a series of *contained (child) objects* (for example, cases on a pallet).
- A *QuantityEvent* is an event that happens to a specified number of objects all having the same type, but where the individual instances are not identified. For example a QuantityEvent could report that an event happened to 200 boxes of widgets, without identifying specifically which boxes were involved. Quantity Events can serve as a bridge between RFID systems and legacy inventory systems that do not identify individuals items.
- A *TransactionEvent* describes the association or disassociation of physical objects to a business transaction.

There is no graphical user interface (GUI) associated with the event service.

For a detailed description the event service, see "Event Service" on page 3-1.

Master Data Service

The master data service associates business-context information with event data. Master data is created and maintained through the Master Data Administration Console. The data is stored in the master data repository and is mapped to associated events in response to requests from the data exchange and reporting services. For the data exchange service, master data is not returned as part of a query, but it can be specified in a query as a means for filtering the results. For the reporting service, master data is returned as part of a report.

For a detailed description the master data service, see "Master Data Service" on page 4-1.

Data Exchange Service

In order to realize the business benefits of RFID, trading partners must be able to exchange data. For example, a consumer goods manufacturer can achieve better execution of, and visibility into, sales promotions by tagging individual items, but only if the manufacturer can see data from both its own readers and data generated by the retailer's readers.

The data exchange service makes data available to clients. The service is subscription-based, where a subscription consists of a set of query parameters, a destination, and a schedule. Each time a query is run, the results are sent to the destination specified for that subscription. Subscriptions are created and managed through the Query Subscription Administration Console.

For a detailed description of the data exchange service, see "Data Exchange Service" on page 5-1.

Reporting Service

The reporting service provides four types of reports, which are accessed via the reporting service console:

- **History**: Where and when were assets last seen, and in what context.
- **Missing Assets**: Which assets have not been seen since the specified time, and optionally, at the specified location.
- **Dwell Time**: How long have assets remained at the specified location.
- **Transit Time**: How long does it take for assets to move from one location to another.

There is no programmatic access to the reporting service; the reporting service console is the only way to generate reports.

Because the reporting service has its own manual, it receives only a brief mention in this one. See *Generating WebLogic RFID Enterprise Server Reports* for a full description of the reporting service.

Services: Data Flow, GUI, and Name Space Summary

Table 2-1 provides a brief summary of each service's data input and output formats; and the name, URL, and online help for each service that has a GUI.

Table 2-2 lists the XSDs for the EPCIS components.

Table 2-1 Services Summary: Data Input, Data Output, and GUIs

Service	Data Input	Data Output	GUI / Login / Online Help
Event	**Internal:** None **External:** Event data from edge as encapsulated XML via HTTP, JMS queue, or SOAP. Store data in event repository.	**Internal:** To EPCIS query engine and reporting engine. **External:** None	None
Master Data	**Internal:** None **External:** Master data from GUI (manually entered or file upload). Store data in master data repository.	**Internal:** To reporting service. **External:** None	Master Data Administration Console `http://<host>:7001/masterdataadminconsole` Online help is integrated with console, and also available at: ***Master Data Administration Console Help***
Data Exchange	**Internal**: Data from event engine and master data engine. **External:** Subscription information and query parameters from GUI. Store data in subscriptions repository	**Internal:** To internal applications via internal APIs. **External:** Export query results in XML format via standard notification mechanisms: HTTP, JMS, AS2, console, or file.	Query Subscription Administration Console `http://<host>:7001/epcis-console` Online help is integrated with console, and also available at: ***Query Subscription Administration Console Online Help***
Reporting	**Internal:** Data from event engine and master data engine. **External:** Query parameters from GUI.	**Internal:** None **External:** Display in GUI or export to file (CSV or XML format).	Reporting Service Console `http://<host>:7001/enterprise-reports` Online help is the Generating *WebLogic RFID Enterprise Server Reports* manual, which is accessible from the console and the Internet.

When you install RFID Enterprise Server on Microsoft Windows, a link to the RFID Enterprise Server Home page is added to the Start menu. To access the Home page directly, open a Web browser and follow this link: `http://host:port/enterprise/`.

Table 2-2 Schema Filenames and Target Namespace for Events and Master Data

Component and XSD or WSDL Filename	Target Namespace
EPCIS Events `EPCIS.xsd`	`urn:epcglobal:epcis:xsd:1`
Event - Capture `EPCIS-capture.xsd`	`http://www.bea.com/ns/rfid/enterprise/epcis-capture/xsd/2.0/`
Event - Query `EPCIS-query.xsd`	`urn:epcglobal:epcis-query:xsd:1`
Master Data `EPCIS-masterdata.xsd`	`urn:epcglobal:epcis-masterdata:xsd:1`
EPCIS WSDL `EPCIS.wsdl`	`urn:epcglobal:epcis-wsdl:xsd:1`

CHAPTER 3

Event Service

In the RFID Enterprise Server, there are two types of RFID-related 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 interface. An example of event data is “At Time T, Object X was observed at Location L.” Event data is captured at the edge and sent (SOAP, JMS queue, or HTTP) either to an RFID Enterprise Server for storage in the event repository or directly to an application.

  Event data grows in quantity as more business is transacted, and refers to things that happen at specific moments in time. There is no mechanism to delete or modify an event; events are retracted or corrected via a new event whose business meaning is to rescind or amend the effect of a prior event.

- Master data is additional data that provides the necessary context for interpreting event data. It is available for filtering a query through the EPCIS Query Interface, and available as part of a report through the Reporting Service. An example of master data is “Location L refers to the distribution center located at 123 Elm Street, Anytown, US." Master data does not come from the edge; it is created and maintained through the Master Data Administration Console, which is provided with WebLogic RFID Enterprise Server.

  Master data does not grow merely because more business is transacted. It is not typically tied to specific moments in time and it provides interpretation for elements of event data. For more information about the master data service, see “Master Data Service” on page 4-1.

Each event is of a specific event type, and contains event fields which provide information about the event. The following sections describe event types and fields; the last section provides an

example showing how event data is formatted as XML when sending data from the edge to the enterprise:

- "Event Types" on page 3-5
- "Event Fields" on page 3-11
- "Events XML Example" on page 3-20

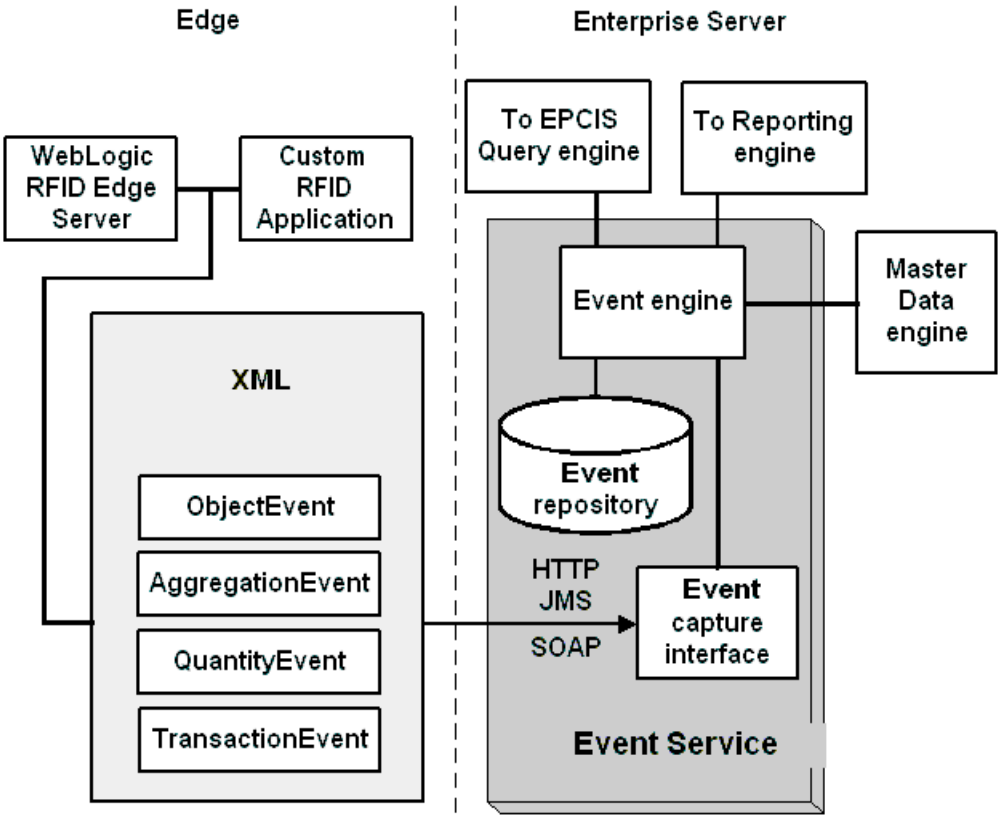
Figure 3-1 shows that the event service receives RFID event data from an RFID Enterprise Server or a custom application, stores the data in the event repository, and interacts with the engines in other subsystems to integrate the data with master data and make the data available to the reporting and data exchange services.

Note: There is no GUI associated with the event service because the service does not require interactive input from administrators or users. Once its database and input mechanisms are configured for the host WebLogic Server, the event service accepts properly formatted event data input from the edge, stores the data in the event repository, and interacts with the other services without further ado.

For information about sending event data from WebLogic RFID Edge Server to WebLogic RFID Enterprise Server, see the Edge Server documentation at:

http://e-docs.bea.com/rfid/edge_server/docs21/workflow_reference/index.html

Figure 3-1 Event Service



The remainder of this introductory section contains Listing 3-1, an XML example of data for a single event, and Figure 3-2 which shows how some event fields reference master data associated with that event.

Listing 3-1 shows data for an event of type `ObjectEvent` formatted as XML. This type of information is what the RFID Enterprise Server receives from the edge and stores in the event repository for later retrieval via a query.

Listing 3-1 ObjectEvent Data in XML Format

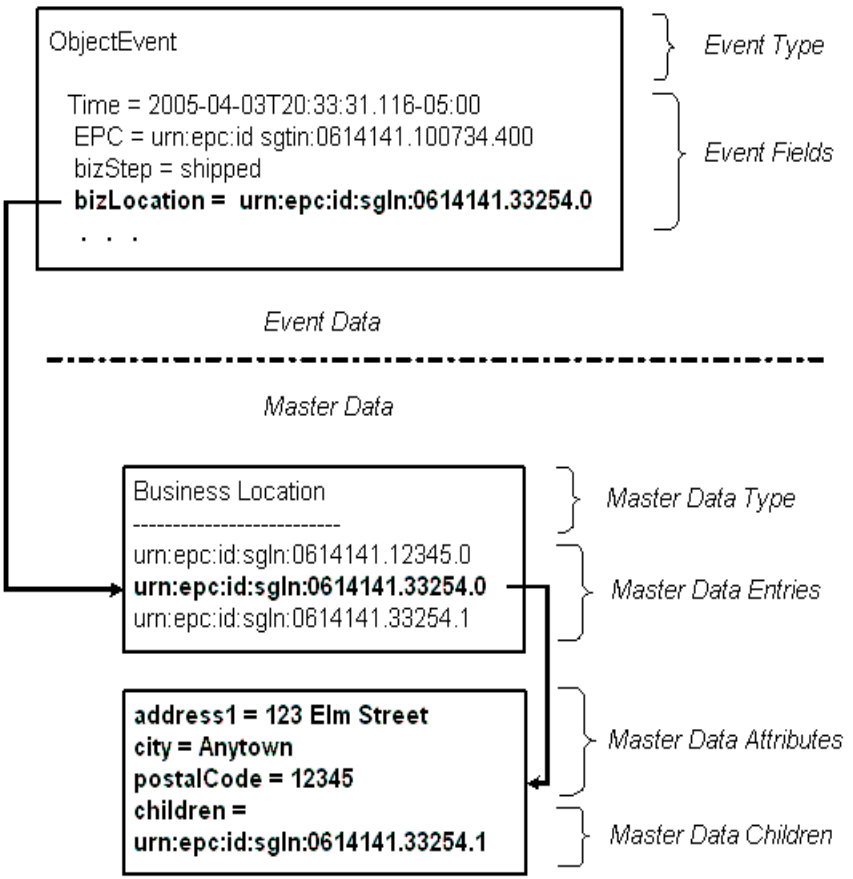
```
<ObjectEvent>
  <eventTime>2006-04-03T20:33:31.116-05:00</eventTime>
  <epcList>
    <epc>urn:epc:id:sgtin:0614141.100734.400.3</epc>
  </epcList>
  <action>OBSERVE</action>
  <bizStep>urn:epcglobal:epcis:bizstep:fmcg:shipped</bizStep>
  <disposition>urn:epcglobal:epcis:disp:fmcg:unknown</disposition>
  <readPoint>
    <id>urn:epc:id:sgln:0614141.07347.1234</id>
  </readPoint>
  <bizLocation>
    <id>urn:epc:id:sgln:0614141.33254.0</id>
  </bizLocation>
  <bizTransactionList>
    <bizTransaction type="urn:epcglobal:fmcg:btt:po">
       http://transaction.acme.com/po/1234
    </bizTransaction>
  </bizTransactionList>
</ObjectEvent>
```

- The `eventTime` indicates that the event occurred on April 3, 2006.
- Because there is only one URN in the `epcList`, the event contains data only for a single item.
- According to the `action`, the item was merely observed at this location.
- The elements `bizStep`, `disposition`, `readPoint`, `bizLocation`, and `bizTransactionList` are examples of event fields that have corresponding master data types; in this case: Business Step, Disposition, Read Point, Business Location, Business Transaction, and Business Transaction Type.

Figure 3-2 illustrates the relationship between events and master data. The name of an event field (`bizLocation`) can provide a reference to a master data type (Business Location), and the value of an event field (`urn:epc:id:sgln:0614141.33254.0`) can reference a master data entry that points to the master data attributes associated with that URN. This use of event fields and values as indexes into a master data repository is the simple mechanism that associates business-context

information with a specific event. (Children are discussed in **"Master Data Service" on page 4-1.**)

Figure 3-2 Event and Master Data



Event Types

There are five event types, one generic event and four subclasses. Table 3-1 describes these event types.

Table 3-1 Event Types

Event Type	Description
`EPCISEvent`	A generic base class for all event types; it provides date and time fields. You cannot capture or query an EPCISEvent; you can capture and query the following four events, which inherit these two fields.
`ObjectEvent`	An event that happened to one or more entities denoted by EPCs.
`AggregationEvent`	An event that happened to one or more EPC-denoted entities that are physically aggregated (constrained to be in the same place at the same time, as when cases are aggregated to a pallet).
`QuantityEvent`	An event concerned with a specific number of objects all having the same type, but where the individual instances are not identified. For example a quantity event could report that an event happened to 200 boxes of widgets, without identifying specifically which boxes were involved. Quantity events can serve as a bridge between RFID systems and legacy inventory systems that do not identify individuals items.
`TransactionEvent`	An event in which one or more EPC-denoted entities become associated or disassociated with an identified business transaction.

The following sections provide descriptions of these event types and show the relevant portions of the XML schema (XSD):

- "EPCISEvent" on page 3-6
- "ObjectEvent" on page 3-7
- "AggregationEvent" on page 3-8
- "QuantityEvent" on page 3-9
- "TransactionEvent" on page 3-10

EPCISEvent

`EPCISEvent` is a common base type for all EPCIS events, providing date and time information.

```
<xsd:complexType name="EPCISEventType" abstract="true">
  <xsd:sequence>
```

```
    <xsd:element name="eventTime" type="xsd:dateTime"/>
    <xsd:element name="recordTime" type="xsd:dateTime" minOccurs="0"/>
    …
  </xsd:sequence>
  <xsd:anyAttribute processContents=”lax”/>
</xsd:complextype>
```

For descriptions of the fields, see Table 3-2.

ObjectEvent

Logically, an `ObjectEvent` pertains to a single object identified by an EPC. However, you can specify more than one EPC in an `epcList` when the remaining `ObjectEvent` data applies to all the EPCs in the list. The ability to provide more than one EPC in an `ObjectEvent` is simply a shorthand to reduce the volume of data in the case where several EPCs experience an identical event.

In an `ObjectEvent`, no relationship among the EPCs is implied by their appearing in the same `ObjectEvent`, other than the coincidence of them all being captured with identical information. By contrast, an `AggregationEvent` or `TransactionEvent` conveys an implicit association among the EPCs in the event.

```
<xsd:complexType name="ObjectEventType">
  <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:sequence>
      <xsd:anyAttribute processContents="lax"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

For field descriptions, see Table 3-2.

AggregationEvent

An `AggregationEvent` describes events related to objects that have been physically aggregated. In such an event, there is a set of *contained* objects that have been aggregated within a *containing* entity which identifies the physical aggregation itself. An example of an aggregation is where cases are loaded onto a pallet and carried as a unit; the pallet may well have a tag, but the *containing* entity is not the physical pallet, it is the aggregate of the pallet and all the cases.

Because an `AggregationEvent` indicates aggregations among physical objects, the children are identified by EPCs. However, the parent entity is identified by an arbitrary URI (which may or may not be an EPC) because the parent is not necessarily a physical object that is separate from the aggregation itself.

For example, it is common in many manufacturing operations to create a load several steps before an EPC is assigned to that load. In such situations, an internal tracking number is assigned at the time the load is created, and this number is used up to the point of shipment. In terms of events, this flow could be modeled as separate `AggregationEvents`:

- When the load is created, an `AggregationEvent` uses the internal tracking number as the parent identifier, expressed as a URI, with `action` equal to `ADD`.
- At the point of shipment, the load is assigned an SSCC code, which is an EPC. A second `AggregationEvent` with `action` equal to `ADD` is generated at the time the SSCC is assigned. (The first association could also be invalidated by generating an `AggregationEvent` with `action` equal to `DELETE` at this time.)

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

For descriptions of the fields, see Table 3-2.

Note: In order to reset or to definitively set an aggregation, send an `AggregationEvent` with action DELETE and no children, followed by an `AggregationEvent` with action ADD and the definitive set of children.

QuantityEvent

A `QuantityEvent` captures an event that pertains to a specified quantity of an object class, but where the individual instances are not identified. A `QuantityEvent` is useful, for example, to report inventory levels of a product. In addition, quantity events can serve as a bridge between RFID systems and legacy inventory systems that do not identify individuals items.

Because an `epcClass` always denotes a specific packaging unit (for example, 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 the master data for that object class.

```
<xsd:complexType name="QuantityEventType">
  <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 minOccurs="0" name="bizTransactionList"
            type="epcis:BusinessTransactionListType"/>
          …
        </xsd:sequence>
        <xsd:anyAttribute processContents="lax"/>
      </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

For descriptions of the fields, see Table 3-2.

TransactionEvent

A `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 can 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.

```
<xsd:complexType name="TransactionEventType">
  <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:sequence>
      <xsd:anyAttribute processContents="lax"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

For descriptions of the fields, see Table 3-2.

Event Fields

Table 3-2 lists each event field by its XSD name and type, describes the field's function, and, if a master data type is associated with the event field name, provides the master data type name. If the description is lengthy, the table provides a short description followed by a reference to a section containing a detailed explanation.

Table 3-2 Event Field Descriptions

Event Field (XSD name & type hierarchy)	Description, XML Example, and Associated Master Data Type
`action` (`ActionType`) (string)	One of `ADD`, `OBSERVE`, or `DELETE`. See "action" on page 3-18. XML example: `<action>ADD</action>` Associated master data type: none

`bizLocation` (`BusinessLocationType`) (`anyURI`)	The location where an object is assumed to be following an EPCIS event until the object is reported to be at a different `bizLocation` by a subsequent EPCIS event.
	A `bizLocation` consists of an `id` (`anyURI`).
	XML example:
	`<bizLocation>`
	`<id>urn:epc:id:sgln:0614141.33254.0</id>`
	`</bizLocation>`
	Associated master data type: Business Location

`bizStep` (`BusinessStepIDType`) (`anyURI`)	The business context of an event: what business process step was taking place that caused the event to be captured? An example is an identifier that means "shipped."
	XML example:
	`<bizStep>`
	`urn:epcglobal:epcis:bizstep:fmcg:shipped`
	`</bizStep>`
	Associated master data type: Business Step

Table 3-2 Event Field Descriptions

Event Field (XSD name & type hierarchy)	Description, XML Example, and Associated Master Data Type

`bizTransactionList`
`<bizTransaction type=>`
`(anyURI)`
	A list describing one or more business transactions associated with the event. Each transaction (`bizTransaction`) element has an optional `type` attribute.
	XML example:
	`<bizTransactionList>`
	`<bizTransaction`
	`type="urn:epcglobal:fmcg:btt:po">`
	`http://transaction.acme.com/po/54847`
	`</bizTransaction>`
	`</bizTransactionList>`
	Associated master data types: Business Transaction, Business Transaction Type
	See "bizTransactionList" on page 3-19.

`childEPCs`
`(EPCListType)`
`(EPC)`
	A list of EPCs (listEPC) represented in "pure identity" URI form as defined in Section 4.1 of the *EPC Tag Data Standard Version 1.1 rev 1.27*.
	XML example:
	`<childEPCs>`
	`<epc>`
	`urn:epc:id:sgtin:0614141.107346.2017`
	`</epc>`
	`</childEPCs>`
	Associated master data type: none

Table 3-2 Event Field Descriptions

Event Field (XSD name & type hierarchy)	Description, XML Example, and Associated Master Data Type

`disposition`
`(DispositionIDType)`
`(anyURI)`
	The business state of an object. An example is an identifier that means "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.
	XML example:
	`<disposition>`
	`urn:epcglobal:epcis:disp:fmcg:unknown`
	`</disposition>`
	Associated master data type: Disposition

`epcClass`
`(epcClassType)`
`(anyURI)`
	Classes of trade items. Any EPC whose structure incorporates the concept of object class can be referenced as an `epcClass`. "EPCClass" on page 3-20 provides the standards for SGTIN EPCs.
	XML example:
	`<epcClass>`
	`urn:epc:idpat:sgtin:0652642.*.*`
	`</epcClass>`
	Associated master data type: EPCCLass

`epcList`
`(EPCListType)`
`(EPC)`
	EPCs represented in "pure identity" URI form as defined in Section 4.1 of the *EPC Tag Data Standard Version 1.1 rev 1.27*.
	XML example:
	`<epcList>`
	`<epc>`
	`urn:epc:id:sgtin:0614141.107346.2017`
	`</epc>`
	`</epcList>`
	Associated master data type: none

Table 3-2 Event Field Descriptions

Event Field (XSD name & type hierarchy)	Description, XML Example, and Associated Master Data Type

`eventTime` `(dateTime)`	The date and time at which the capturing application asserts the event occurred.
	XML example:
	`<eventTime>`
	`2006-04-03T20:33:31.116-05:00`
	`</eventTime>`
	Associated master data type: none

`parentID` `(ParentIDType)` `(anyURI)`	The identifier of the parent of the EPCs in `epcList` (a parent ID can be a URI that is not a URN). When the parent identifier is an EPC, it is represented in "pure identity" URI form as defined in Section 4.1 of the *EPC Tag Data Standard Version 1.1 rev 1.27*.
	XML example:
	`<parentID>`
	`urn:epc:id:sgtin:0614141.107346.2017`
	`</parentID>`
	Associated master data type: none

`quantity` `(int)`	The number of objects within the class described by this event.

Table 3-2 Event Field Descriptions

Event Field (XSD name & type hierarchy)	Description, XML Example, and Associated Master Data Type

`readPoint` `(ReadPointType)` `(anyURI)`	The location that identifies the most specific place at which an EPCIS event took place. The `readPoint` for an event is determined by the 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. Conceptually, `readPoint` identifies "how or where the EPCIS event was detected."
	
	A `readPoint` consists of an `id` (`anyURI`).
	
	XML example:
	
	`<readPoint>`
	`  <id>`
	`  urn:epc:id:sgln:0614141:07347.1234`
	`  </id>`
	`</readPoint>`
	
	Associated master data type: Read Point

`recordTime` `(dateTime)`	The date and time at which this event was recorded by an event repository. This field is always omitted when an event is presented to the capture interface, and always present when an event is retrieved from the query interface. The `recordTime` information is useful when interpreting results of standing queries (subscriptions). XML example: none Associated master data type: none

Table 3-3 shows which event fields are used by each event type. The **R** and **O** keys in the event type columns indicate whether a field is required (**R**) for that event or optional (**O**). The absence of a key means that the field is not used by that event type; for example, `action` is not used by `QuantityEvent`.

Table 3-3 Event Types: Required and Optional Fields

R = Required Field O = Optional Field	ObjectEvent	AggregationEvent	QuantityEvent	TransactionEvent
action	R	R		R
bizLocation	O	O	O	O
bizStep	O	O	O	O
bizTransactionList	O	O	O	R
childEPCs		R		
disposition	O	O		O
epcClass			R	
epcList	R			R
eventTime	R	R	R	R
parentID		R		
quantity			R	
readPoint	O	O	O	O
recordTime		[1]		

1. The recordTime field is omitted when an event is captured. It is supplied when an event is queried.

Each event type has event fields that represent the **what**, **when**, **where**, and **why** information for an event:

1. **What** object(s) or other entities are the subject of the event. For example, for an `ObjectEvent` the subject is an `epcList`; for a `QuantityEvent` the subject is an `epcClass` and `quantity`; for a `TransactionEvent`, the subject is a `bizTransactionList`.
2. **When** did the event occur: the date and time (the `eventTime`).
3. **Where** did the event occur: for example, `readPoint` and `bizLocation`.

4. **Why** did the event occur (the business context): for example, `bizStep` and `disposition`.

action

An `action` indicates how an event relates to the lifecycle of the entity.

The `Action` type is an enumerated type having three possible values: `ADD`, `OBSERVE`, or `DELETE`. Table 3-4 describes what these `Action` values mean for the three event types that include an `action` field as event data.

Table 3-4 Action Field Values

	ObjectEvent	AggregationEvent	TransactionEvent
`ADD`	EPCs in `epcList` are commissioned.	EPCs in `childEPCs` are aggregated to parent (added to the running tabulation for this aggregation). A parent identifier is required.	`EPCs in epcList` are associated with a transaction. Either with a new transaction or additional EPCs are added to existing transaction.
`OBSERVE`	Simple observation: neither a commission nor a decommission event.	Simple observation: there is no guarantee that all children are observed, or that the parent is either observed or known. A parent identifier is optional.	EPCs are confirmed as continuing to be associated with the transaction.
`DELETE`	EPCs in `epcList` are decommissioned.	EPCs in `childEPCs` are disaggregated. If `childEPCs` is omitted, all children are disaggregated. A parent identifier is required.	EPCs in `epcList` are disassociated from the transaction. If `epcList` is omitted, all children are disassociated.

For example, an `AggregationEvent` captures events related to 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 indicates 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

business step, bizStep, which identifies the specific business process step in which the action took place.

bizTransactionList

A bizTransactionList identifies a particular business transaction, such as a specific purchase order. An ObjectEvent, QuantityEvent, or AggregationEvent may include bizTransactionList; a TransactionEvent must include bizTransactionList.

A business transaction is described by a pair of identifiers, bizTransaction and type. Each bizTransactionList contains one or more bizTransaction elements; each bizTransaction element may include a type attribute. Both bizTransaction and type are of type anyURI:

```
<bizTransactionList>
  <bizTransaction type=anyURI>anyURI</bizTransaction>
  ...
</bizTransactionList>
```

If type is omitted, no information is available about the type of business transaction apart from what is implied by the value of the bizTransaction field itself.

The following XSD fragments show the relationships between bizTransactionList, bizTransaction, and type:

```
<!-- bizTransactionList -->
<xsd:element name="bizTransactionList"
   type="epcis:BusinessTransactionListType" minOccurs="0"/>

<!-- bizTransactionListType -->
<xsd:complexType name="BusinessTransactionListType">
  <xsd:sequence>
    <xsd:element name="bizTransaction"
       type="epcis:BusinessTransactionType" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<!-- BusinessTransactionType -->
<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>
```

An example of a business transaction is:

```
<bizTransactionList>
  <bizTransaction type=”urn:epcglobal:fmcg:btt:po”>
     http://transaction.acme.com/po/54847
  </bizTransaction>
</bizTransactionList>
```

The master data type associated with `bizTransaction` is Business Transaction. The master data type associated with `type` is Business Transaction Type.

EPCClass

`EPCClass` entries denote classes of trade items. Any EPC whose structure incorporates the concept of object class can be referenced as an EPCClass.

When an `EPCClass` represents a class of SGTIN EPCs, the following form is required:

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

- `CompanyPrefix` is an EAN.UCC Company Prefix (including leading zeros)
- `ItemRefAndIndicator` consists of the indicator digit of a GTIN followed by the digits of the item reference of a GTIN.

An `EPCClass` entry 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.

Events XML Example

The following document shows the results of a query that includes two events, one where an object (represented by EPC `urn:epc:id:sgtin:0614141.107346.2017`) travels from one location (represented by `urn:epc:id:sgln:0614141.33254.0`) to another (`urn:epc:id:sgln:0614141.33482.0`). The values for business step, disposition, and business transaction are all values chosen collaboratively by the implementers of the EPCIS data exchange solution. Because standards in certain industries are still under development, business partners may choose to agree on a set of values that satisfy their business needs.

```
<?xml version="1.0" encoding="UTF-8"?>
<epcis:EPCISDocument xmlns:epcis="urn:epcglobal:epcis:xsd:1"
 schemaVersion="1" creationDate="2006-07-11T11:30:47.0Z">
 <EPCISBody>
   <EventList>
      <ObjectEvent>
          <eventTime>2006-04-03T20:33:31.116-05:00</eventTime>
          <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:transit
          </disposition>
          <readPoint>
              <id>urn:epc:id:sgln:0614141.07347.124</id>
          </readPoint>
          <bizLocation>
              <id>urn:epc:id:sgln:0614141.33254.0</id>
          </bizLocation>
          <bizTransactionList>
              <bizTransaction type="urn:epcglobal:fmcg:btt:po">
                  http://transaction.acme.com/po/54847
              </bizTransaction>
          </bizTransactionList>
      </ObjectEvent>
      <ObjectEvent>
          <eventTime>2006-04-04T20:33:31.116-05:00</eventTime>
          <epcList>
              <epc>urn:epc:id:sgtin:0614141.107346.2017</epc>
          </epcList>
          <action>OBSERVE</action>
          <bizStep>urn:epcglobal:epcis:bizstep:fmcg:received</bizStep>
          <disposition>
              urn:epcglobal:epcis:disp:fmcg:processing
          </disposition>
```

```
            <readPoint>
                <id>urn:epc:id:sgln:0614141.07473.384</id>
            </readPoint>
            <bizLocation>
                <id>urn:epc:id:sgln:0614141.33482.0</id>
            </bizLocation>
            <bizTransactionList>
                <bizTransaction type="urn:epcglobal:fmcg:btt:po">
                    http://transaction.acme.com/po/54847
                </bizTransaction>
            </bizTransactionList>
        </ObjectEvent>
    </EventList>
 </EPCISBody>
</epcis:EPCISDocument>
```

CHAPTER 4

Master Data Service

As stated in "Event Service" on page 3-1, and reiterated here for reference, the EPCIS-related services deal with two types 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 interface. An example of event data is "At Time T, Object X was observed at Location L." Event data is captured at the edge and sent (SOAP, JMS, or HTTP) either to an RFID Enterprise Server for storage in the event repository or directly to an application.

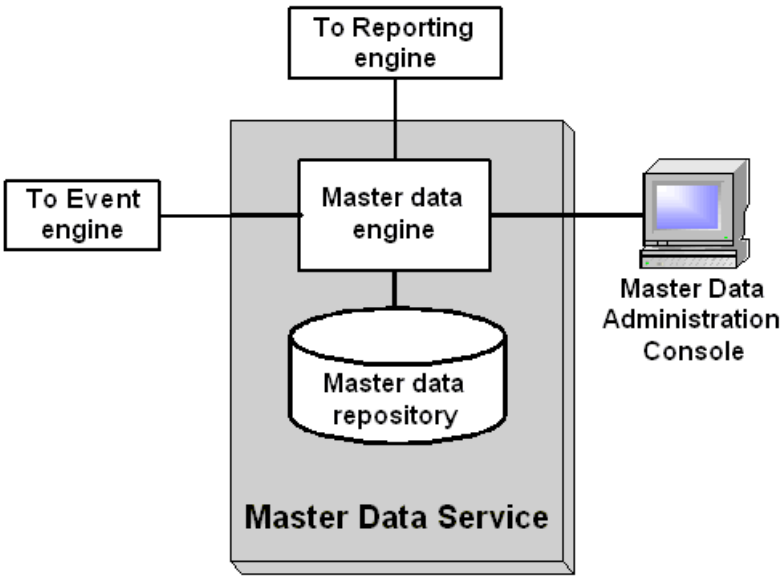
  Event data grows in quantity as more business is transacted, and refers to things that happen at specific moments in time. There is no mechanism to delete or modify an event; events are retracted or corrected via a new event whose business meaning is to rescind or amend the effect of a prior event.

  For more information about the event service, and an introductory figure that shows the relationship between event data and master data, see "Event Service" on page 3-1.

- Master data is additional data that provides the necessary context for interpreting event data. It is available for filtering a query through the EPCIS Query Interface, and available as part of a report through the Reporting Service. An example of master data is "Location L refers to the distribution center located at 123 Elm Street, Anytown, US." Master data does not come from the edge; it is created and maintained through the Master Data Administration Console, which is provided with WebLogic RFID Enterprise Server.

  Master data does not grow merely because more business is transacted. It is not typically tied to specific moments in time and provides interpretation for elements of event data.

Figure 4-1 Master Data Service



Master data is business-context information that is associated with event data by the reporting service and by the data exchange service (queries for events can include parameters that filter query results based on master data).

Master data is not present in the captured event data in the WebLogic RFID Enterprise Server 2.0 release; it is created and maintained separately. The relationship between event data and master data is that event data can be used as an index into relevant master data. For example, consider an event that contains following event type and value:

```
<bizLocation>
  <id>urn:epc:id:sgln:0614141.33254.0</id>
</bizLocation>
```

Unless you already know the answer, the URN representation provides data but not much information. However, when the URN is used as a master data Business Location entry which reference a set of master data attributes, you can associate useful information with the event. This association is shown in Figure 3-2, "Event and Master Data," on page 3-5, where `urn:epc:id:sgln:0614141.33254.0` references the following master data attributes:

- address1 = 123 Elm Street
- city = Anytown
- postalCode = 12345
- children = urn:epc:id:sgln:0614141.33254.1

Thus the ability to create master data and associate it with information contained in events provides an efficient mechanism for associating additional business information with tag data.

Note: Events and master data have separate XML schemas (XSDs). (The master data schema is used when returning information from a SimpleEventQuery.) The XML types mentioned in this section refer to those used in the master data XSD, not the event data XSD.

The following sections examine master data relationships:

- "Master Data Types" on page 4-3
- "Master Data Entries, Attributes, and Children" on page 4-4
- "Master Data XML Example" on page 4-7

Master Data Types

A master data type is a definition for master data entries of that type.

Each master data type defines a set of attributes and their data types; for example, as shipped, the Business Location type defines an attribute named `address1`, whose URI is `urn:epcglobal:epcis:mda:name`, and whose data type is `String` ("Master Data Entries, Attributes, and Children" on page 4-4 provides tables that list the attributes for master data types.)

Types and the attributes associated with them are extensible: you can use the WebLogic RFID Enterprise Server Master Data Administration Console to create and modify master data types, entries, and attributes.

Table 4-1 shows the master data type name for each event type that has a master data type associated with it.

Table 4-1 Field Events and Associated Master Data Types

Field Event Name	Associated Master Data Type and Formal URI Name
`bizLocation`	Business Location `URI="urn:epcglobal:epcis:vtype:BusinessLocation"`
`bizStep`	Business Step `URI="urn:epcglobal:epcis:vtype:BusinessStep"`

`bizTransactionList`	`bizTransaction`: Business Transaction
	`URI="urn:epcglobal:epcis:vtype:BusinessTransaction"`
	`type`: Business Transaction Type
	`URI="urn:epcglobal:epcis:vtype:BusinessTransactionType"`

`disposition`	Disposition `URI="urn:epcglobal:epcis:vtype:Disposition"`
`epcClass`	EPC Class `URI="urn:epcglobal:epcis:vtype:EPCClass"`
`readPoint`	Read Point `URI ="urn:epcglobal:epcis:vtype:ReadPoint"`

Master Data Entries, Attributes, and Children

A master data *entry* is a concrete instance of a master data type. You can create as many entries as you need of each master data type. Each entry has the same set of attributes defined for its master data type, but where the master data type defines the data type for each attribute, an entry's *attributes* contain the real business-context information associated with a business's operations.

For example, the master data type Business Location defines an attribute `address1` of type `String`. A master data Business Location entry could define `address1` as `123 Elm Street`.

In addition to attributes that provide concrete business information, an entry's attribute list can contain zero or more *children*. A child is the identifier of another master data entry within the same master data type. You can create a hierarchy of entries when it makes business sense to do so. For example, assume three Business Location entries that reference "Acme Corp. Retail Store #3," "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 entry is the parent and the latter two entries are children.

- Acme Corp. Retail Store #3
  - Acme Corp. Retail Store #3 Backroom
  - Acme Corp. Retail Store #3 Sales Floor

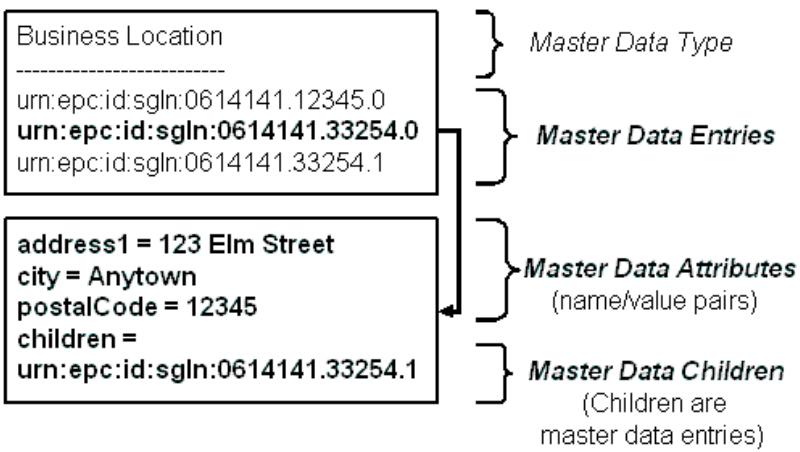
This ability to create a hierarchical parent/child relationship among entries is useful for establishing informative relationships among entries. For example, you have a Business Location entry that represents a building. You create a child entry for each floor in the building. Each floor entry has child entries that represent discrete areas such as “billing office,” “store room,” “receiving,” “storage,” etc.

One additional relationship that you can establish between entries is that an entry can be the child of more than one parent. This allows for more than one way of grouping entries; for example, locations could be grouped both by geography and by function.

Note: To prevent a circular relationship, an entry cannot become its own descendant via the parent/child relationship. For example, a child entry cannot specify a parent entry as one of its children.

In Figure 4-2, `urn:epc:id:sgln:0614141.33254.1` is a child of the master data entry `urn:epc:id:sgln:0614141.33254.0`.

Figure 4-2 Master Data Entries, Attributes, and Children



See “Master Data XML Example” on page 4-7 for examples of attributes and children.

WebLogic RFID Enterprise Server 2.0 provides the following pre-defined attributes for each business type:

- Table 4-2, “Business Location and Read Point Attributes,” on page 4-6
- Table 4-3, “Business Step, Business Transaction, Business Transaction Type, Disposition, and EPC CLass Attributes,” on page 4-7

The data type for all attributes in these tables is String.

Table 4-2 Business Location and Read Point Attributes

Name	URI
Name	urn:epcglobal:epcis:mda:name
Description	urn:epcglobal:epcis:mda:description
Address1	urn:epcglobal:epcis:mda:address1
Address2	urn:epcglobal:epcis:mda:address2
Adress3	urn:epcglobal:epcis:mda:address3
City	urn:epcglobal:epcis:mda:city
State or province	urn:epcglobal:epcis:mda:stateProvince
Country	urn:epcglobal:epcis:mda:country
Postal code	urn:epcglobal:epcis:mda:postalCode
FMCG GLN	urn:epcglobal:fmcg:mda:gln
FMCG site sublocation type	urn:epcglobal:fmcg:mda:sslt
FMCG site sublocation attributes	urn:epcglobal:fmcg:mda:sslta
FMCG site location extension	urn:epcglobal:fmcg:mda:sle

Table 4-3 Business Step, Business Transaction, Business Transaction Type, Disposition, and EPC CLass Attributes

Name	URI
Name	urn:epcglobal:epcis:mda:name
Description	urn:epcglobal:epcis:mda:description

Master Data XML Example

In the following example, keep in mind the following mappings:

- A master data type is represented by the `<Vocabulary>` element
- A master data entry is represented by the `<VocabularyElement>` element
- A master data attribute is represented by the `<attribute>` element
- A master data child is represented by the `<children>` element

```
<EPCISMasterDataBody>
    <VocabularyList>
      <Vocabulary type="urn:epcglobal:epcis:vtype:BusinessLocation">
        <VocabularyElementList>
          <VocabularyElement id="urn:epc:id:sgln:0614141.33254.0">
            <attribute id="urn:epcglobal:fmcg:mda:sslt:warehouse"/>
          </VocabularyElement>
          <VocabularyElement id="urn:epc:id:sgln:0614141.33482.0">
            <attribute id="urn:epcglobal:fmcg:mda:sslt:retail"/>
            <attribute id="urn:epcglobal:fmcg:mda:address">
              <acme:Address
                xmlns:acme="http://acme.com/types">
                <Street>100 Nowhere Street</Street>
                <City>Omaha</City>
                <State>NB</State>
                <Zip>70475</Zip>
              </sample:Address>
            </attribute>
            <children>
```

```
                <id>urn:epcglobal:fmcg:ssl:0614141.33482.201</id>
              </children>
            </VocabularyElement>
           <VocabularyElement id="urn:epcglobal:fmcg:ssl:0614141.33482.201">
              <attribute id="urn:epcglobal:fmcg:mda:sslt:201"/>
            </VocabularyElement>
          </VocabularyElementList>
        </Vocabulary>
      </VocabularyList>
  </EPCISMasterDataBody>
```

CHAPTER 5

Data Exchange Service

The primary function of the Data Exchange Service is to configure query subscriptions and send query results formatted in XML to trading partners via HTTP, JMS, and AS2.

The Data Exchange Service provides several access points to the Event Repository, allowing the user to perform queries on these events:

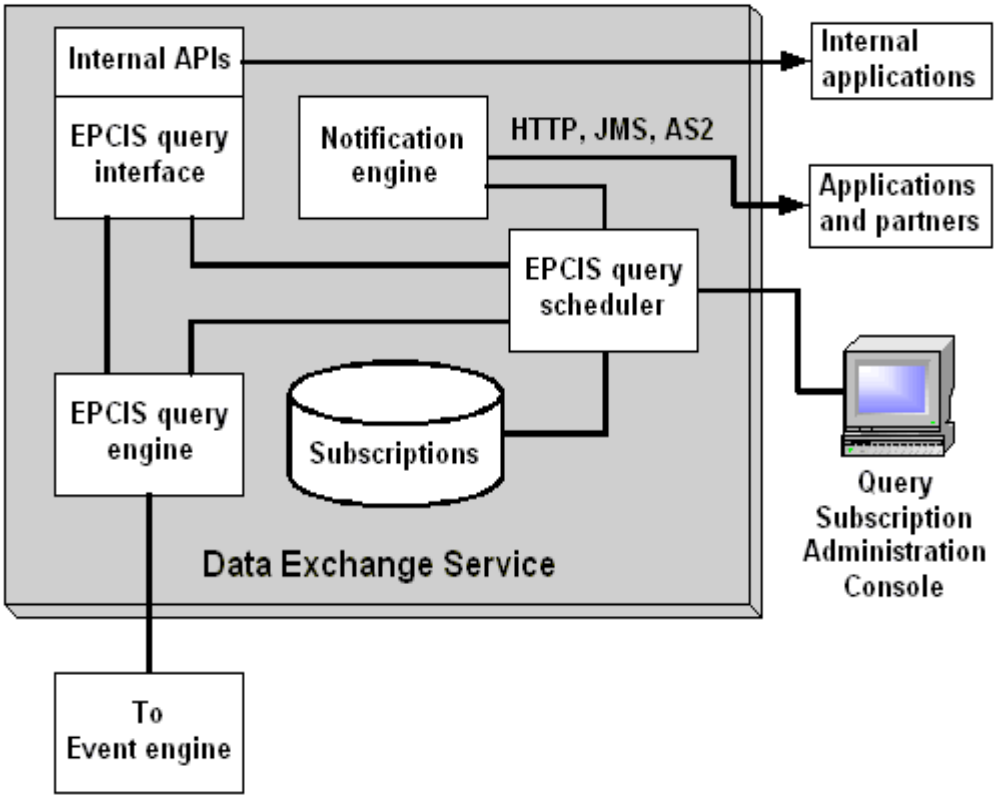
- The Query Subscription Administration Console configures the Data Exchange Service to create queries and periodically deliver query results to designated receivers (HTTP, JMS, and AS2). The Console can also test queries before subscribing, displaying user-readable events in XML format.
- A SOAP Web service allows programmatic queries. This SOAP service is based on the technically complete, but not formally ratified as of October 2006, EPCglobal *EPC Information Services (EPCIS) Version 1.0 Specification*.EPCglobal EPCIS 1.0 Working Draft.

The following sections provide information about query parameters and data security:

- "SimpleEventQuery" on page 5-2
- "Data Security" on page 5-10

Figure 5-1 shows the components in the Data Exchange Service.

Figure 5-1 Data Exchange Service



SimpleEventQuery

A SimpleEventQuery filters the Event Repository by various criteria based on the fields in the events. This type of query is the one configured through the Query Subscription Administration Console and available programmatically through a SOAP service.

A query is defined by a set of query parameters, each of which has a name and a value. For each parameter, Table 5-1 provides a description and the following information:

- **Console Name** is the name used by the Query Subscription Administration Console for the parameter. (The Console uses identical or similar names for parameters, often grouping

similar parameters under a single heading.) Some parameters are only available via the SOAP service.

- **SOAP Name** is the parameter name in a SOAP query definition object (see "Capture and Query APIs" on page 6-1).
- **Value Type** is the data type of the parameter's value to include in a SOAP query definition object (see"Query Interface Parameters and Data Types" on page 6-5).

Table 5-1 SimpleEventQuery Parameter Name/Value Pairs

Parameter	Description
Console Name: Event Type **SOAP Name:** `eventType` **Value Type:** List of String	The result will only include events whose type matches one of the types specified in the parameter value. Each element of the parameter value may be one of the following strings: • `ObjectEvent` • `AggregationEvent` • `QuantityEvent` • `TransactionEvent` 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.
Console Name: After Event Time **SOAP Name:** `GE_eventTime` **Value Type:** Time	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).
Console Name: Before Event Time **SOAP Name:** `LT_eventTime` **Value Type:** Time	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).

Table 5-1 SimpleEventQuery Parameter Name/Value Pairs

Parameter	Description
Console Name: After Record Time **SOAP Name:** `GE_recordTime` **Value Type:** Time	Only events with `recordTime` greater than or equal to the specified value will be returned. If omitted, events are included regardless of their `recordTime`, other than the following limitation: The first time a query is executed for a 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 execution, the only events considered are those whose `recordTime` field is greater than or equal to the time when the query was last executed.
Console Name: Before Record Time **SOAP Name:** `LT_recordTime` **Value Type:** Time	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 limitation based on event record time; See `GE_recordTime`.
Console Name: Action **SOAP Name:** `EQ_action` **Value Type:** List of String	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: `ADD`, `OBSERVE`, or `DELETE`. If omitted, events are included regardless of their `action` field.
Console Name: Business Step **SOAP Name:** `EQ_bizStep` **Value Type:** List of String	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 omitted, events are returned regardless of the value of the `bizStep` field or whether the `bizStep` field exists at all.
Console Name: Disposition **SOAP Name:** `EQ_disposition` **Value Type:** List of String	Analogous to the `EQ_bizStep` parameter, but for the `disposition` field.
Console Name: Read Point **SOAP Name:** `EQ_readPoint` **Value Type:** List of String	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.

Table 5-1 SimpleEventQuery Parameter Name/Value Pairs

Parameter	Description
Console Name: Read Point **SOAP Name:** `WD_readPoint` **Value Type:** List of String	WD is an abbreviation for *with descendants*. 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. If this parameter and `EQ_readPoint` and `readPoint` are omitted, events are returned regardless of the value of the `readPoint` field or whether the `readPoint` field exists at all.
Console Name: Business Location **SOAP Name:** `EQ_bizLocation` **Value Type:** List of String	Analogous to the `EQ_readPoint` parameter, but for the `bizLocation` field.
Console Name: Business Location **SOAP Name:** `WD_bizLocation` **Value Type:** List of String	Analogous to the `WD_readPoint` parameter, but for the `bizLocation` field.
Console Name: Business Transaction **SOAP Name:** `EQ_bizTransaction_type` **Value Type:** List of String	This is not a single parameter, but a family of parameters. If a parameter of this form is specified, the result will only include events that (a) include a `bizTransactionList`; (b) where the business transaction list includes an entry whose type subfield is equal to type extracted from the name of this parameter; and (c) where the `bizTransaction` subfield of that entry is equal to one of the values specified in this parameter.
Console Name: EPC Matches One of the Patterns **SOAP Name:** `MATCH_epc` **Value Type:** List of String	The result will only include events that (a) have an `epc` field (that is, `ObjectEvents` or extension event type that extend `ObjectEvent`); and where (b) the `epc` field matches one of the EPC patterns specified in this parameter. Each element of the parameter list is a pure identity pattern. If this parameter is omitted, events are included regardless of their `epc` field of whether the `epc` field exists.

Table 5-1 SimpleEventQuery Parameter Name/Value Pairs

Parameter	Description
Console Name: Parent ID Matches One of the Patterns **SOAP Name:** `Match_parentID` **Value Type:** List of String **Required**: No	Analogous to `MATCH_epc`, but applies to the `parentID` field of `AggregationEvent` and the `parentID` field of `TransactionEvent`.
Console Name: Child EPC matches one of the Patterns **SOAP Name:** `MATCH_childEPC` **Value Type:** List of String	Analogous to `MATCH_epc`, but applies to the `childEPCs` field of an `AggregationEvent`. An event is a candidate for inclusion if any of the elements of its `childEPCs` match any of the patterns specified in this parameter.
Console Name: EPC Class Matches One of the Patterns **SOAP Name:** `MATCH_epcClass` **Value Type:** List of String	Analogous to `MATCH_epc`, but applies to the `epcClass` field of a `QuantityEvent`.
Console Name: Quantity **SOAP Name:** `EQ_Quantity` **Value Type:** Int	The result will only include events that (a) have a `quantity` field (that is, a `QuantityEvent`); and where (b) the `quantity` field is equal to the specified parameter.
Console Name: Quantity **SOAP Name:** `GT_Quantity` **Value Type:** Int	Analogous to `EQ_quantity`, but includes events whose `quantity` field is greater than the specified parameter
Console Name: Quantity **SOAP Name:** `GE_Quantity` **Value Type:** Int	Analogous to `EQ_quantity`, but includes events whose `quantity` field is greater than or equal to the specified parameter.
Console Name: Quantity **SOAP Name:** `LT_Quantity` **Value Type:** Int	Analogous to `EQ_quantity`, but includes events whose `quantity` field is less than the specified parameter.
Console Name: Quantity **SOAP Name:** `LE_Quantity` **Value Type:** Int	Analogous to `EQ_quantity`, but includes events whose `quantity` field is less than or equal to the specified parameter

Table 5-1 SimpleEventQuery Parameter Name/Value Pairs

Parameter	Description
Console Name: (SOAP only) **SOAP Name:** `EQ_fieldname` **Value Type:** List of String	This is not a single parameter, but a family of parameters. If a parameter of this form is specified, the result will only include events that (a) have a field named *`fieldname`* whose type is either String or a master data type; and where (b) the value of that field matches one of the values specified in this parameter. A *`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.
Console Name: (SOAP only) **SOAP Name:** `EQ_fieldname` **Value Type:** Int, Float, Time	Analogous to `EQ_fieldname`, 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.
Console Name: (SOAP only) **SOAP Name:** `GT_fieldname` **Value Type:** Int, Float, Time	Analogous to `EQ_fieldname`, 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.
Console Name: (SOAP only) **SOAP Name:** `GE_fieldname` **Value Type:** Int, Float, Time	Analogous to `GT_fieldname`.
Console Name: (SOAP only) **SOAP Name:** `LT_fieldname` **Value Type**: Int, Float, Time	Analogous to `GT_fieldname`.
Console Name: (SOAP only) **SOAP Name:** `LE_fieldname` **Value Type:** Int, Float, Time	Analogous to `GT_fieldname`.

Table 5-1 SimpleEventQuery Parameter Name/Value Pairs

Parameter	Description
Console Name: (SOAP only) **SOAP Name:** `EXISTS_`*`fieldname`* Value Type: Void	Analogous to `EQ_`*`fieldname`*, 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`*. Note that the value for this query parameter is ignored.
Console Name: (SOAP only) **SOAP Name:** HASATTR_*`fieldname`* Value Type:	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 master data type; and (b) where the value of that field is a master data entry 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. A *`fieldname`* is the fully qualified name of a field. For a standard field, this is simply the field name; for example, `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.
Console Name: Field with Attribute **SOAP Name:** `EQATTR_`*`fieldname_attrname`* **Value Type:** List of String	This is not a single parameter, but a family of parameters. If a parameter of this form is specified, the result will only include events that (a) have a field named *`fieldname`* whose type is a master data type; and (b) where the value of that field is a master data entry 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. A fieldname is constructed using the same rules as `HASATTR_`*`fieldname`*.
Console Name: Order By **SOAP Name:** `orderBy` **Value Type:** String	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. The value of this parameter MUST 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. If omitted, defaults to `eventTime`.

Table 5-1 SimpleEventQuery Parameter Name/Value Pairs

Parameter	Description
Console Name: Order Direction **SOAP Name:** `orderDirection` **Value Type:** String	If specified and `orderBy` is also specified, specifies whether the results are ordered in ascending or descending sequence according to the key specified by `orderBy`. The value of this parameter MUST be one of `ASC` (for ascending order) or `DESC` (for descending order). If omitted, defaults to `DESC`.
Console Name: Event Count Limit **SOAP Name:** `eventCountLimit` **Value Type:** Int	Results will only include the first N events that match the other criteria, where N is the value of this parameter. The ordering specified by the `orderBy` and `orderDirection` parameters determine the meaning of "first" for this purpose. If omitted, all events matching the specified criteria will be included in the results. This parameter 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.

Console Name: Max Event Count
SOAP Name: `maxEventCount`
Value Type: Int
	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` is raised instead of a normal query result.
	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.
	The value for `MaxEventCount` is configurable via an `<env-entry>` element in:
	`BEA_HOME/user_projects/applications/`*`your-domain`*`/epcis-lib.ear/epcis-ejb.jar/META-INF/ejb-jar.xml`
	The element is:
	`<env-entry>`
	`  <env-entry-name>`
	`  `**`maxEPCISEventRecords`**
	`  </env-entry-name>`
	`  <env-entry-type>`
	`  java.lang.String`
	`  </env-entry-type>`
	`  <env-entry-value>`**`1000`**`</env-entry-value>`
	`</env-entry>`

Data Security

A user who has one of the roles that provides the ability to create subscriptions (`Administrator`, `epcis_admin`, or `rfid_admin`) is allowed full access to all events in the Event Repository. For this reason, if you have security concerns about access to event data, BEA recommends that only trusted users be given the right to access the Query Subscription Administration Console or the SOAP service, and that query subscriptions are carefully crafted to expose only the desired data to business partners.

CHAPTER 6

Capture and Query APIs

The following sections provide information about the capture and query interfaces.

- "Capture Interface" on page 6-1: The capture API is the simpler of the two, consisting of only one method, `capture()`, which takes a single argument and has no return value. A capture application uses this method to send EPCIS events to an Enterprise Server for insertion into an EPCIS database.
- "Query Interface" on page 6-3: The query API provides several methods for extracting information from an EPCIS database.

The methods described in these sections constitute the public SOAP API for the WebLogic RFID Enterprise Server. For XSD and WSDL information, see Table 2-2, "Schema Filenames and Target Namespace for Events and Master Data," on page 2-7.

Note: In all API calls, EPCs should be provided in Pure Identity format; if EPCs are specified in tag format, they will be stored in tag format and Pure Identity format.

Capture Interface

The EPCIS Capture Interface provides the ability to capture and persist EPCIS Events. The capture interface is exposed as JMS, HTTP, and SOAP.

The `capture()` method records EPCIS events:

```
capture(event : List<EPCISEvent>) : void
```

The `capture()` method takes one argument, a List of `EPCISEvent`, and returns no results. Because the argument is a list, you can capture one or more EPCIS Events at a time. The events in the following list are those described in "Event Service" on page 3-1:

- `EPCISEvent`

  Note: Because `EPCISEvent` is abstract; only events of the following four types can actually be captured.

- `ObjectEvent`
- `AggregationEvent`
- `QuantityEvent`
- `TransactionEvent`

Events can be captured via JMS, SOAP, and HTTP.

Note: For backwards compatibility with 1.x versions of Enterprise Server, Enterprise Server 2.0 supports the `captureEPCISEvent()` method. The difference between these two capture methods is that `capture()` can record multiple EPCIS events at a time while `captureEPCISEvent()` can record only one EPCIS event at a time. For information about `captureEPCISEvent()`, see the *Enterprise Server 1.1 Administrator's Guide* at http://e-docs.bea.com/rfid/rftagaware/pdf/ESAdministratorsGuide1_1.pdf.

All Enterprise Server 1.x capture and query interfaces are supported via a separate Web service in Enterprise Server 2.0. The 1.1 SOAP API is exposed in the 2.0 release. The URL for the 1.1 SOAP service is:

```
http://[host]:[port]/legacyepcis/LegacyEPCIS
```

For SOAP, the inbound-message URL for the WebLogic RFID Enterprise Server 2.0 SOAP service is `http://[host]:[port]/epcis/EPCIS`.

For JMS, three inbound-message queues are provided in WebLogic RFID Enterprise Server 2.0:

- `EPCISMessages` (1.1 messages)
- `EPCISCapture` (2.0 messages)
- `EPCISFailedMessages` (failed messages of all types)

The `EPCISMessages` queue used in version 1.1 is also available in 2.0. Therefore applications that sent messages to this queue in Enterprise Server 1.1 can send continue to send messages to the same queue in Enterprise Server 2.0 without any modifications. This queue should operate seamlessly, assuming that security configurations are compatible between the sending and receiving systems.

The `EPCISCapture` queue is a new queue for capturing messages formatted for Enterprise Server 2.0.

The `EPCISFailedMessages` queue can hold messages of either type. When an error occurs during capture, a message is automatically moved to this queue after the number of retries is exceeded.

For HTTP, the inbound-message URL is `http://[host]:[port]/EPCISServlet`.

Query Interface

The query interface supports querying the EPCIS event repository by constructing a query and a set of associated query parameters. Query parameters include time constraints, event types, event fields, action, disposition, and master data name/value attributes. Both synchronous and asynchronous queries are supported through polling and subscription mechanisms. The query interface is exposed as a SOAP Service defined by a WSDL and set of XSD files.

The following sections provide information about the query interface:

Query Interface Methods

The query interface provides two basic types of queries for event information: those that poll for an immediate response from the Event Repository, and those that register a query subscription for periodic delivery of information from the Event Repository.

There are several methods for querying the EPCIS database:

- `getQueryNames() : List  // of names`
- `getStandardVersion() : String`
- `getSubscription (subscriptionID : String) : Subscription`
  `// BEA Extension`
- `getSubscriptionIDs(queryName : String) : List  // of Strings`
- `getVendorVersion() : String`

- `poll(queryName : String, params : QueryParams) : QueryResults`
- `subscribe(queryName : String, params : QueryParams, dest : URI, controls : SubscriptionControls, subscriptionID : String)`

  Note: The query subscription administration console uses the `subscribe()` method.

- `unsubscribe(subscriptionID : String)`

Table 6-1 describes the query methods. "Query Interface Parameters and Data Types" on page 6-5 describes the `queryName` parameter, and the `QueryParams` and `SubscriptionControls` data types.

Table 6-1 Query Methods

Method	Description
`getQueryNames`	Return a List containing the pre-defined query name: `SimpleEventQuery`.
`getSubscription`	Given a `subscriptionID`, return a `Subscription` (BEA extension). A `Subscription` contains: • `subscriptionID` • `notificationURI` • `queryName`
`getSubscriptionIDs`	Return a List of active subscription IDs: all subscription IDs currently subscribed to the specified named query. See "queryName" on page 6-6.
`getStandardVersion`	Returns the version of the EPCIS specification to which this implementation conforms. This method always returns `1.0`.
`getVendorVersion`	Returns a string that identifies the version of any vendor extensions this implementation provides. This method always returns `http://version.connecterra.com/epcis/1`.
`poll`	Poll for synchronous query response. 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 embedded in the definition of the query. See "queryName" on page 6-6, "QueryParams" on page 6-6, and "Query Interface Results" on page 6-12.

Table 6-1 Query Methods

Method	Description
`subscribe`	Subscribe to a Query for asynchronous notification. Registers a subscriber for a previously defined query having the specified name. • `queryName`: a string specifying the name of the query. See "queryName" on page 6-6. • `params` : `QueryParams`: the values used for any named parameters embedded in the definition of the query. See "QueryParams" on page 6-6. • `dest` : `URI`: a URI-specified destination where results from the query are to be delivered. See "Notification URIs" on page 6-10. • `controls` : `SubscriptionControls`: controls how the subscription is to be processed. In particular, it specifies the conditions under which the query is to be invoked; for example, specifying a periodic schedule. See "SubscriptionControls" on page 6-7. • `subscriptionID`: an arbitrary string that is copied into every response delivered to the specified destination, and otherwise not interpreted by the EPCIS service. A client might use the `subscriptionID` to identify from which subscription a given result was generated, especially when several subscriptions are made to the same destination.
`unsubscribe`	Unsubscribe from an existing query. Removes a previously registered subscription having the specified `subscriptionID`.

Query Interface Parameters and Data Types

This section describes the `queryName` parameter and the data types used by query methods (excluding such well-known types as List or String):

- "queryName" on page 6-6 (used by the `poll` and `subscribe` methods)
- `params` : "QueryParams" on page 6-6 (used by the `poll` and `subscribe` methods)
- `controls` : "SubscriptionControls" on page 6-7 (used by the `subscribe` method)
- `schedule` : "QuerySchedule" on page 6-8 (used by `SubscriptionControls`)
- `initialRecordTime` : `Time` (see "SubscriptionControls" on page 6-7)
- `dest` : `URI` ("Notification URIs" on page 6-10, used by the `subscribe` method)

queryName

WebLogic RFID Enterprise Server 2.0 supports the following query name:

- `SimpleEventQuery`

Therefore, when you use `poll()` or `subscribe()`, the String you specify as the `queryName` must be `SimpleEventQuery`. In addition, `getQueryNames()` will return only this name.

"Data Exchange Service" on page 5-1 describes this query and provides tables of the `QueryParam` name/value pairs you can use to define a query request.

QueryParams

`QueryParams` is the data type for the `params` argument used by the `poll`() and `subscribe()` methods.

A `QueryParams` instance is a set of `QueryParam` name/value pairs, where the names correspond to parameter names in the query, and the values are the specific values to be used for that invocation of (`poll`) or subscription to (`subscribe`) the query.

```
<xsd:complexType name="QueryParam">
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="value" type="xsd:anyType"/>
  </xsd:sequence>
</xsd:complexType>
```

Table 6-2 lists the valid XML types you can use when specifying a value.

Table 6-2 XML Types for value Elements

Parameter Type	XML type for `value` element
Int	`xsd:integer`
Float	`xsd:double`
Time	`xsd:dateTime`
String	`xsd:string`
List of String	`epcisq:ArrayOfString`

For `poll()` and `subscribe()`, the valid set of name/value pairs are those supported by `SimpleEventQuery`.

See "SimpleEventQuery" on page 5-2 for detailed descriptions of valid name/value pairs

Note: Although a `QueryParam` is a simple name/value pair, there can be a large matrix of possible pairs that you can specify when constructing a single `params` argument; therein lies both the power and the complexity of creating subscriptions and queries. For example, `SimpleEventQuery` recognizes more than 30 name/value pairs.

Enterprise Server provides the Query Subscription Administration Console, which is the mechanism you use for creating subscription-based queries.

SubscriptionControls

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

Table 6-3 describes the fields of a `SubscriptionControls` instance.

Table 6-3 SubscriptionControls

Argument	Description
Name: `schedule` **Data Type:** QuerySchedule **Required:** No	Defines the periodic schedule on which the query is to be executed. See "QuerySchedule" on page 6-8. You must specify `schedule` or `trigger`, but not both.
Name: `trigger` **Data Type:** URI **Required:** No	Specifies a triggering event known to the EPCIS service that will trigger execution of this query. (Available trigger URIs are service-dependent.) You must specify `trigger` or `schedule`, but not both.

Table 6-3 SubscriptionControls

Argument	Description
Name: `initialRecordTime` **Data Type:** Time **Required:** No	Specifies a time that determines which events are considered when processing a standing query for the first time. If omitted, defaults to the time at which the subscription is created. 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 subsequent execution of the query, the only events considered are those whose `recordTime` field is greater than or equal to the time when the query was last executed.
Name: `reportIfEmpty` **Data Type:** Boolean **Required:** Yes	If true, a `QueryReports` instance is always sent to the subscriber when the query is executed. If false, a `QueryReports` instance is sent to the subscriber only when the results are non-empty.

QuerySchedule

If specified, a `QuerySchedule` defines a periodic schedule for query execution for a specific subscription. 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 conforms 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”
```

Table 6-4 QuerySchedule

Argument	Description
Name: `second` **Data Type:** String **Required:** No	Specifies that the query time must have a matching seconds value. The range for this parameter is 0 through 59, inclusive.
Name: `minute` **Data Type:** String **Required:** No	Specifies that the query time must have a matching minute value. The range for this parameter is 0 through 59, inclusive.
Name: `hour` **Data Type:** String **Required:** No	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.
Name: `dayOfMonth` **Data Type:** String **Required:** No	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.)
Name: `month` **Data Type:** String **Required:** No	Specifies that the query time must have a matching month value. The range for this parameter is 1 through 12, inclusive.
Name: `dayOfWeek` **Data Type:** String **Required:** No	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.

The following `QuerySchedule` examples show how various combinations of elements are interpreted:

```
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.

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

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

```
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.”

```
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.

Notification URIs

The following notification URIs can be used as the `dest` parameter to the `subscribe` method

- “HTTP” on page 6-10
- “Applicability Statement 2 (AS2)” on page 6-11

In all delivery mechanisms, the content is transferred in XML format as specified by the EPCIS XML Schema definition

HTTP

The HTTP notification URI provides for delivery of query results in XML via the HTTP protocol using the POST operation.

An HTTP notification URI has the two following forms:

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

The following table defines the terms used in the HTTP form.

host	The DNS name or IP address of the host where the receiver is listening for incoming HTTP connections.
`port`	The TCP port on which the receiver is listening for incoming HTTP connections. If omitted, the default port for HTTP is 80.
`remainder-of-URL`	The URL to which an HTTP POST operation will be directed.

The message payload is an `EPCISQueryDocument` instance whose `EPCISBody` element contains a `StandingQueryResults` element. For more information, see "Query Interface Results" on page 6-12.

Applicability Statement 2 (AS2)

The AS2 notification URI provides for delivery of query results in XML via AS2.

An AS2 notification URI has the following form:

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

The `remainder-of-URI` identifies a specific AS2 communication profile to be used by the EPCIS Service to deliver information to the subscriber. 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 that wants 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.

The AS2 message payload is an `EPCISQueryDocument` instance whose `EPCISBody` element contains a `StandingQueryResults` element. For more information, see "Query Interface Results" on page 6-12. For information on configuring AS2 as a destination for a query subscription, see Configuring Enterprise Server to Send Query Subscription Results to an AS2 Destination in *Installing WebLogic RFID Enterprise Server*.

Query Interface Results

`QueryResults` is the return type for the `poll` method and for standing queries requested by the `subscribe` method.

- The `poll` method returns a `PollResult` element of type `OnDemandQueryResults`
- The `subscribe` method returns a `StandingQueryResults` element of type `StandingQueryResults`.

Both results include a `resultsBody` element of type `QueryResultsBody`. The following XSD code fragments illustrate this shared use of `QueryResultsBody`:

poll:

```
<xsd:element name="PollResult" type="epcisq:OnDemandQueryResults"/>
   ...
<xsd:complexType name="OnDemandQueryResults">
  <xsd:sequence>
    <xsd:element name="resultsBody" type="epcisq:QueryResultsBody"/>
   ...
  </xsd:sequence>
</xsd:complexType>
```

subscribe:

```
<xsd:element name="StandingQueryResults"
      type="epcisq:StandingQueryResults"/>
<xsd:complexType name="StandingQueryResults">
  <xsd:sequence>
    <xsd:element name="queryName" type="xsd:string"/>
    <xsd:element name="subscriptionID" type="xsd:string"/>
    <xsd:element name="resultsBody" type="epcisq:QueryResultsBody"/>
   ...
  </xsd:sequence>
</xsd:complexType>
```

QueryResultsBody:

```
<xsd:complexType name="QueryResultsBody">
  <xsd:choice>
    <xsd:element name="eventList" type="epcis:EventListType"/>
    <xsd:element name="vocabularyList" type="epcismd:VocabularyListType"/>
```

```
    ...
  </xsd:choice>
</xsd:complexType>
```

Thus, both methods return a `QueryResultsBody` (with a subscription you also receive the `queryName` and the `subscriptionID`). In essence, a query returns either a list of events or a list of master data information.

Event information is in an `eventList` element of type `EventListType`, which contains zero or more instances of the supported event types:

```
<xsd:complexType name="EventListType">
   <xsd:choice maxOccurs="unbounded">
        <xsd:element name="ObjectEvent" type="epcis:ObjectEventType"
             maxOccurs="unbounded"/>
        <xsd:element name="AggregationEvent"
             type="epcis:AggregationEventType" maxOccurs="unbounded"/>
        <xsd:element name="QuantityEvent" type="epcis:QuantityEventType"
             maxOccurs="unbounded"/>
        <xsd:element name="TransactionEvent"
             type="epcis:TransactionEventType" maxOccurs="unbounded"/>
        <xsd:element name="extension"
             type="epcis:EPCISEventListExtensionType"/>
        <xsd:any namespace="##other" processContents="lax"/>
   </xsd:choice>
</xsd:complexType>
```

Master data information is in a `vocabularyList` element of type `VocabularyListType`, which eventually devolves to zero or more instances of a `VocabularyElementType`:

```
<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:sequence>
    <xsd:attribute name="id" type="xsd:anyURI" use="required"/>
    <xsd:anyAttribute processContents="lax"/>
  </xsd:complexType>
```

```
<xsd:complexType name="AttributeType">
   ...
    <xsd:attribute name="id" type="xsd:anyURI" use="required"/>
   ...
</xsd:complexType>

<xsd:complexType name="IDListType">
   ...
    <xsd:element name="id" type="xsd:anyURI" minOccurs="0"
         maxOccurs="unbounded"/>
   ...
</xsd:complexType>
```

Query Interface Exceptions

Table 6-5 lists possible exceptions for the following methods:

- getQueryNames
- getSubscription
- getSubscriptionIDs
- poll
- subscribe
- unsubscribe

Table 6-6 describes the exceptions.

Table 6-5 Query Method Exceptions

Query Method	Exception
getQueryNames	SecurityException ImplementationException
getSubscription	NoSuchNameException SecurityException SecurityException
getSubscriptionIDs	NoSuchNameException SecurityException ImplementationException

Table 6-5 Query Method Exceptions

Query Method	Exception
poll	NoSuchNameException QueryParameterException QueryTooComplexException QueryTooLargeException SecurityException ImplementationException
subscribe	NoSuchNameException InvalidURIException DuplicateSubscriptionException QueryParameterException QueryTooComplexException SubscriptionControlsException SubscribeNotPermittedException SecurityException ImplementationException
unsubscribe	NoSuchSubscriptionException SecurityException ImplementationException

In addition, an attempt to execute a standing query might result in a `QueryTooLarge` exception being sent to a subscriber instead of a normal query result. In this case, the `QueryTooLargeException` includes, in addition to the reason string, the query name and the `subscriptionID` as specified in the `subscribe` call that created the standing query.

Table 6-6 Query Exception Descriptions

Exception	Description
DuplicateSubscriptionException	The specified `SubscriptionID` is identical to a previous subscription that was created and not yet unsubscribed.
ImplementationException	Implementation-specific exception.
InvalidURIException	The URI specified for the subscriber cannot be parsed.
NoSuchNameException	The specified query does not exist.

Table 6-6 Query Exception Descriptions

Exception	Description
`NoSuchSubscriptionException`	The specified `SubscriptionID` does not exist.
`QueryParameterException`	One or more query parameters are invalid.
`QueryTooComplexException`	Either the query or the query parameters exceeded the abilities of the implementation.
`SecurityException`	Access control violation.
`SubscriptionControlsException`	Invalid `SubscriptionControls`.
`SubscribeNotPermittedException`	The specified query is not supported with `subscribe`, only `poll`.

Index