# EPC Middleware Standards

(www.epcglobalinc.org/standards)

## Athens Information Technology

# The EPC Architectural Framework

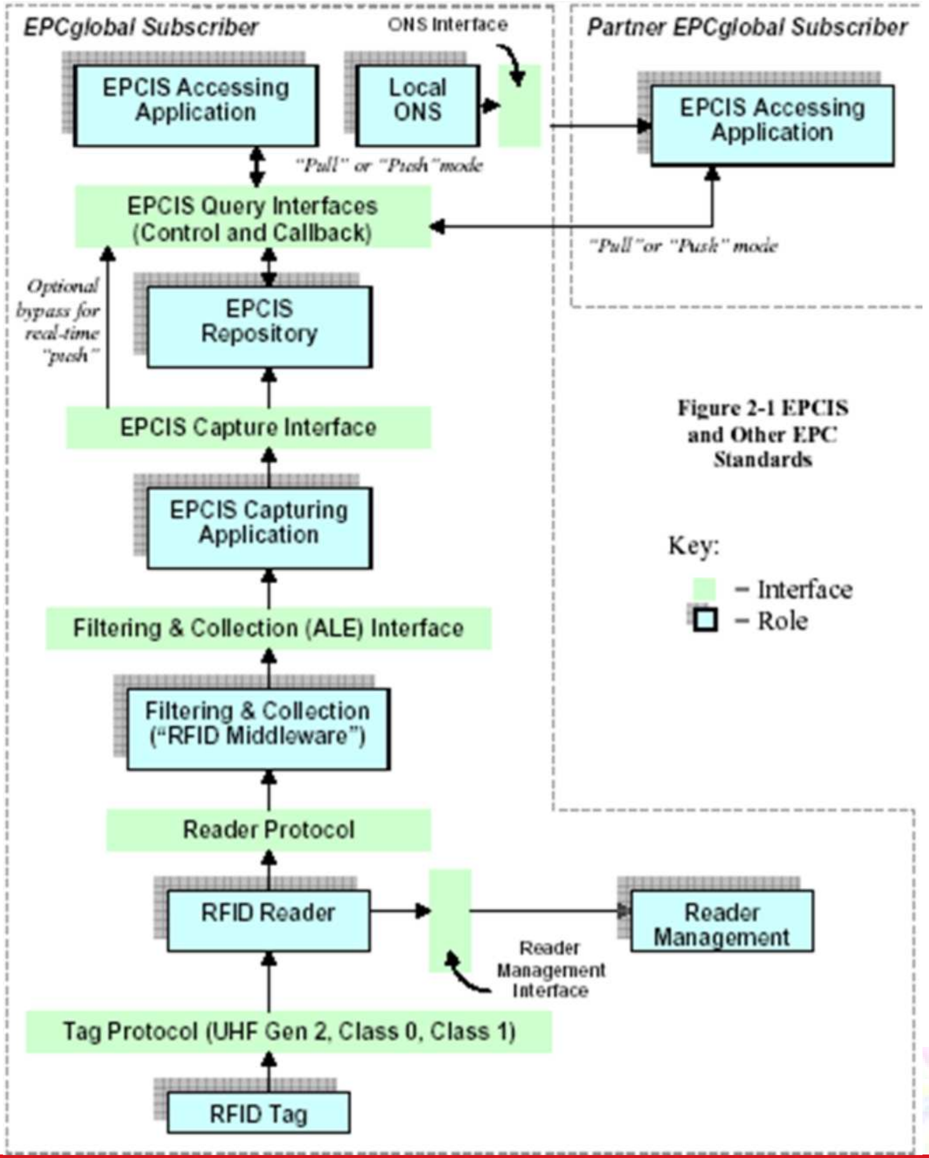
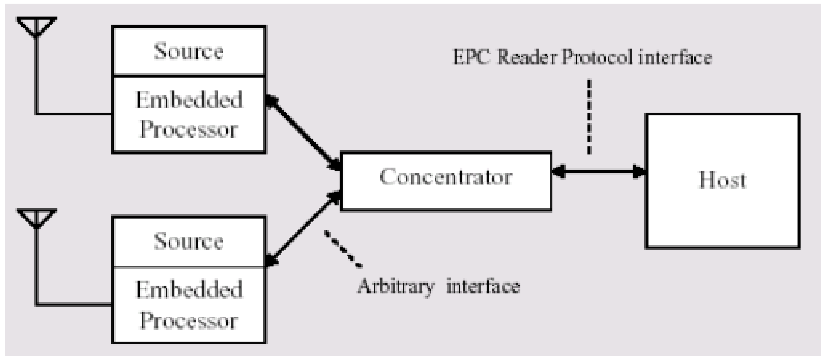
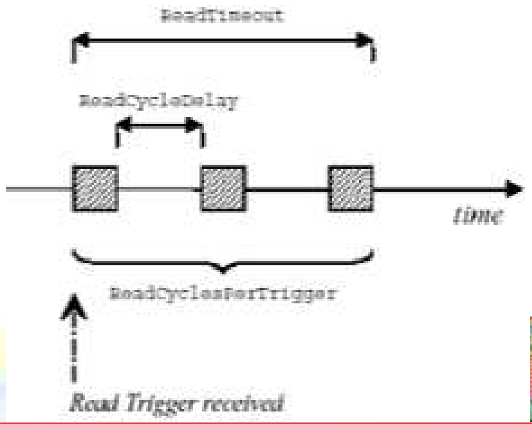

EPCglobal Subscriber
ONS Interface
EPCIS Accessing Application
Local ONS
"Pull" or "Push" mode
EPCIS Query Interfaces (Control and Callback)
Partner EPCglobal Subscriber
EPCIS Accessing Application
"Pull" or "Push" mode
Optional bypass for real-time "push"
EPCIS Repository
EPCIS Capture Interface
EPCIS Capturing Application
Filtering & Collection (ALE) Interface
Filtering & Collection ("RFID Middleware")
Reader Protocol
RFID Reader
Reader Management
Reader Management Interface
Tag Protocol (UHF Gen 2, Class 0, Class 1)
RFID Tag
Key:
🟩 – Interface
□ – Role

Figure 2-1 EPCIS and Other EPC Standards

# The RFID Reader Role – EPC RPv1.1

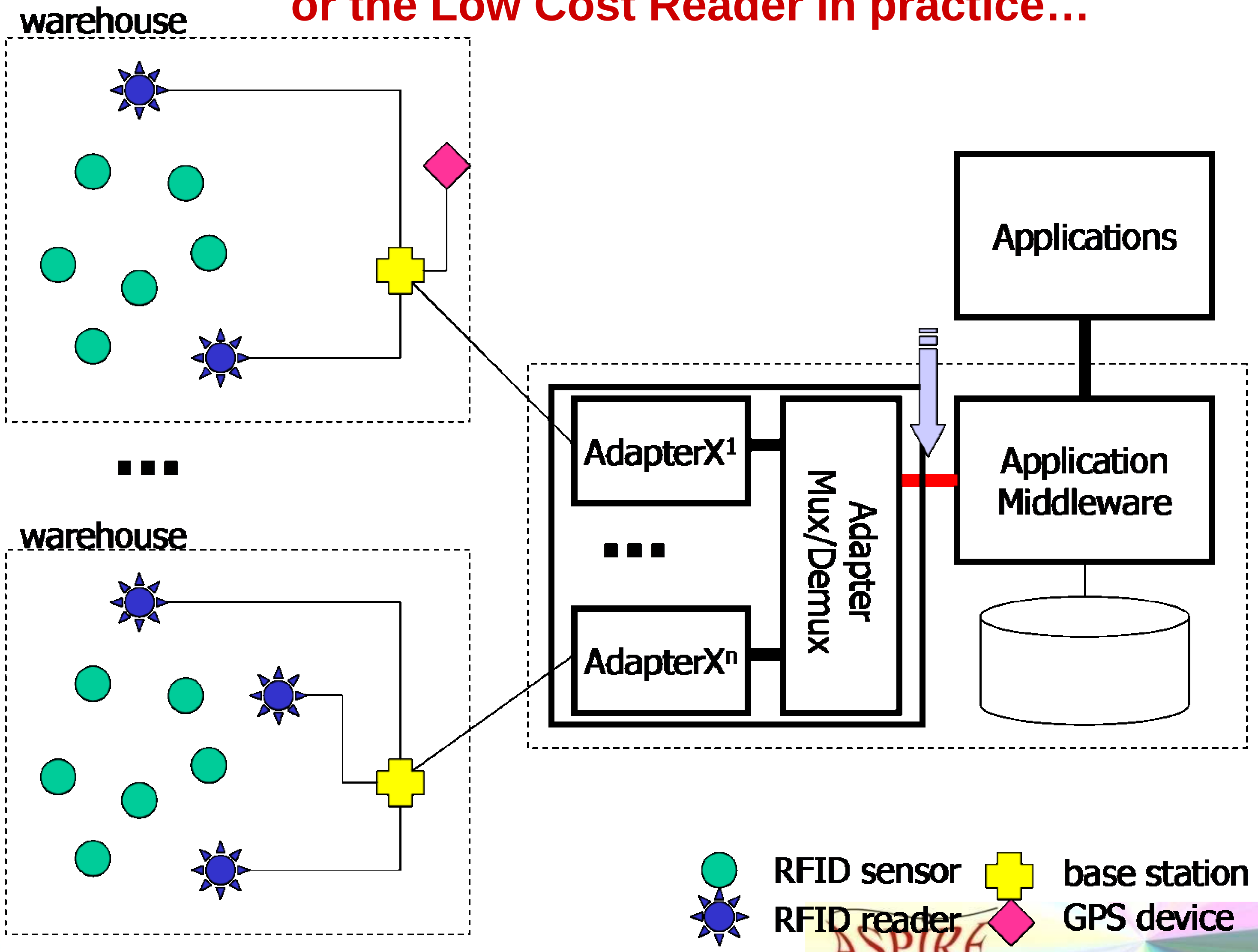
- A RFID Reader is not merely an Interrogator
- An EPC RPv1.1 Reader comprises a set of
  - Readpoints (Antennas or Barcode Scanners)
  - Sources (readpoint concentrators)
- Reader basic functions
  - Detect RFID tags within its EM field (Query function)
  - Repeat Query sessions periodically (on a triggering event basis)
    - EPC RPv1.1 Readers do not depend on external triggers
  - Collect the results of multiple Query session, filter, and report them
  - Implement a Message Transport Binding





## A distributed EPC RPv1.0 RFID Reader

### or the Low Cost Reader in practice…

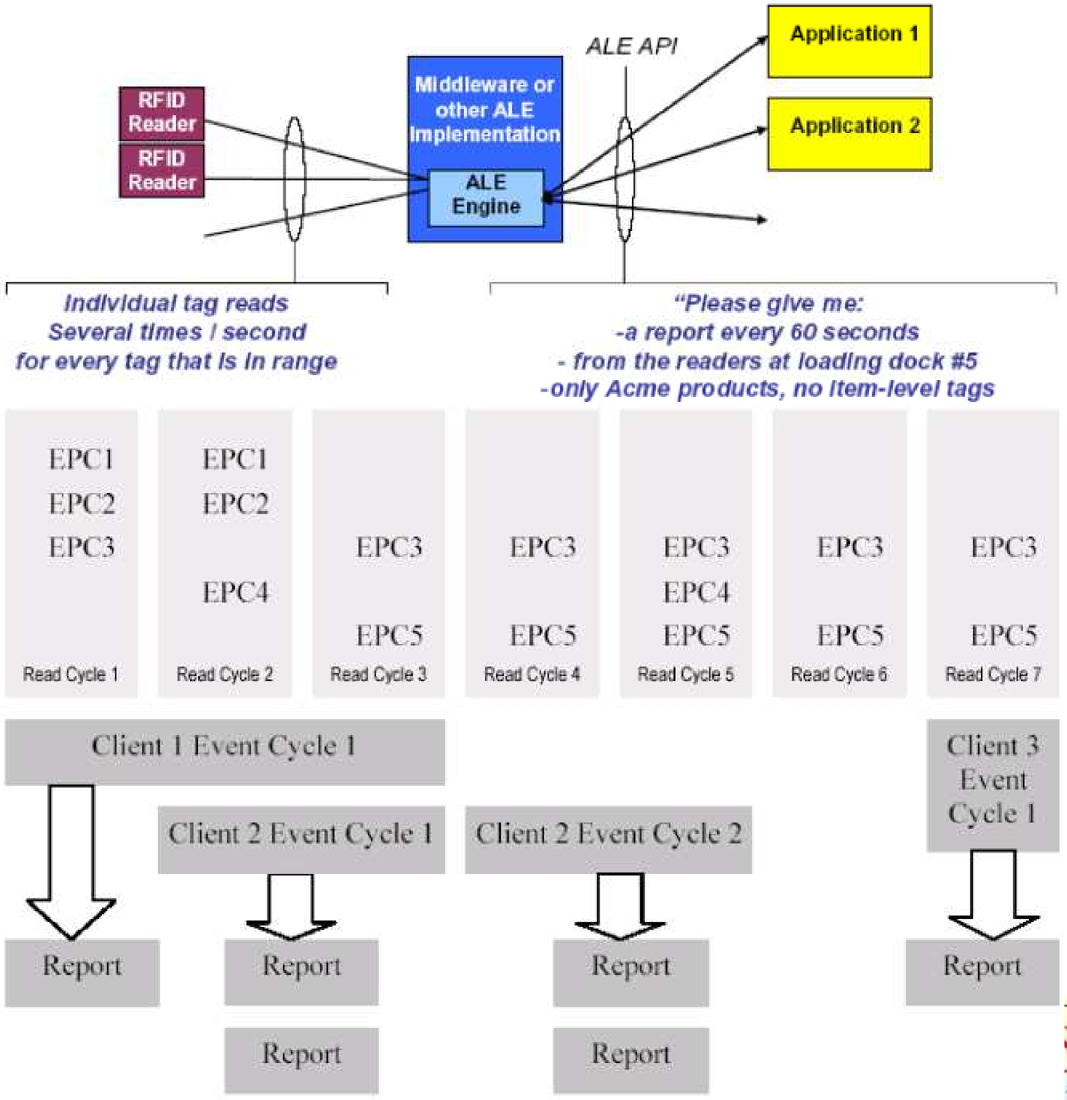


# EPC RPv1.1 Compliance

| Command Set |
|---|
| selectReadPoints(readPoints: ReadPoint[]): void |
| removeReadPoints(readPoints: ReadPoint[]): void |
| addTagSelectors(selectors: TagSelector[]): void |
| removeTagSelectors(selectors: TagSelector[]): void |
| getAllTagSelectors(void): TagSelector[] |
| read(dataSelector: MT_DataSelector, selectors: TagSelector[], triggers: Trigger[]): ReadReport |
| write(data: TagFieldValue[], selectors: TagSelector[], triggers: Trigger[]): void |
| setReadCyclesPerTrigger(cycles : integer): void |
| setMaxReadDutyCycle(dutyCycle:integer): void |
| removeTagFields(tagfields: TagField[]): void |
| getAllTagFields(void): TagField[] |

# Application Level Events

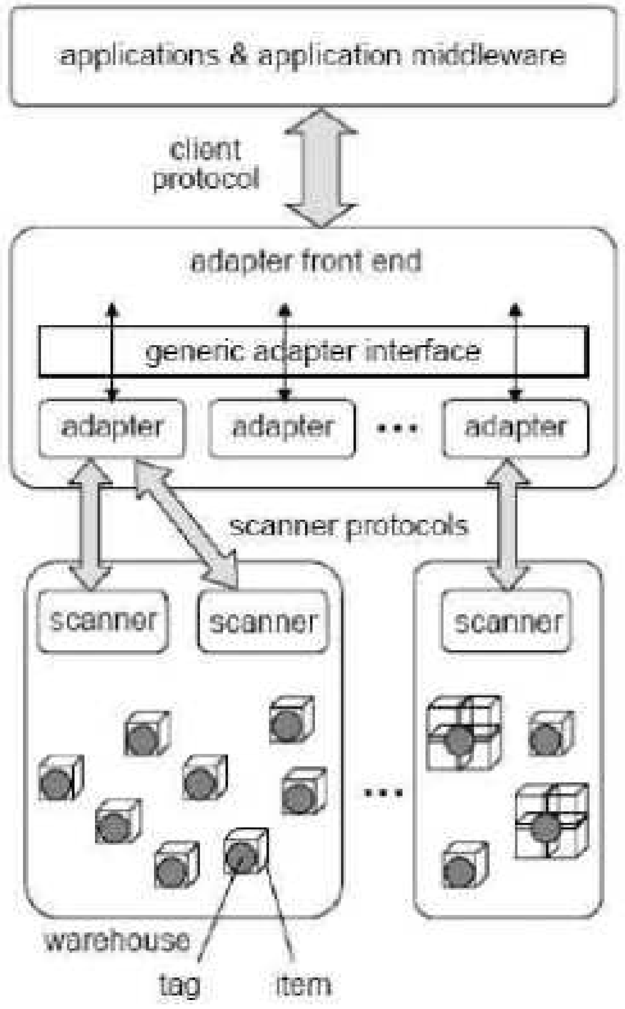
## the name of the game…

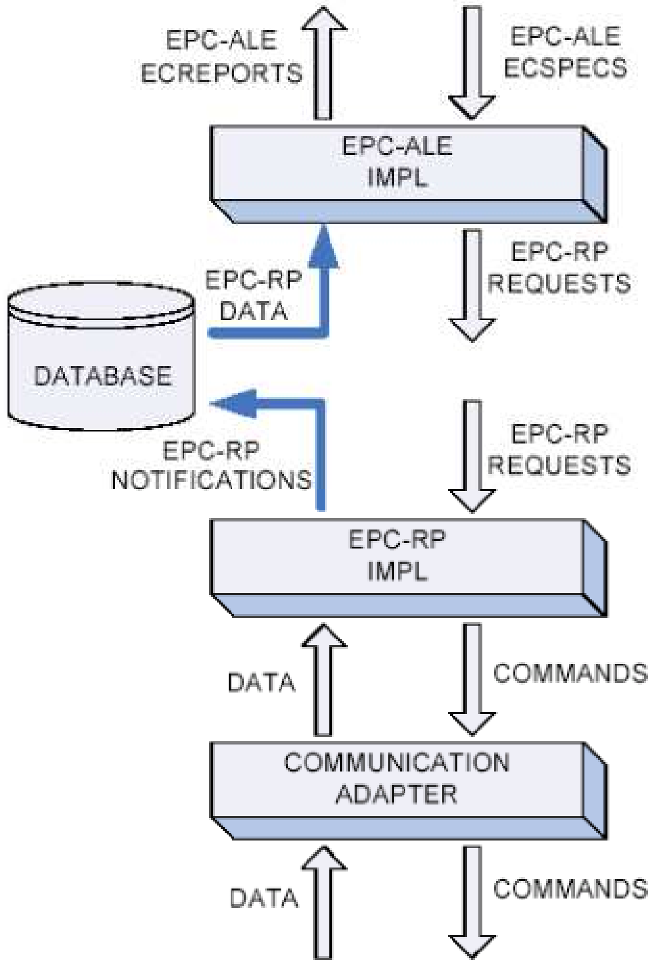


- Data Collection & Filtering
  - Source abstraction
- Space Multiplexing
  - Logical Reader
- Time Multiplexing
  - Event Cycle
- Reads Notifications
  - Elementary data package
- Produces Reports
  - Interoperability with ERP and WM Systems
- The EC Specification

# Using a distributed EPC RPv1.1 Reader

## or the Mobile Warehouse in practice…





# Client Interactions and EPC Application Level Events

## Athens Information Technology

## EPC global architecture network (1)

- Client interactions with EPC data
  - Reading activity and writing activity
- Reading activity
  - Receiving EPCs and related data from one or more data sources such as RFID readers
  - Accumulating data over intervals of time, filtering to eliminate duplicate data and data that are not of interest, and counting and grouping data to reduce the volume of data
  - Reporting in various forms

## EPC global architecture network (2)

- Writing activity
  - Isolating ("singulating") individual data carriers such as RFID Tags through one or more channels such as RFID readers
  - Operating upon the data carriers by writing data, reading data, or performing other operations
  - Reporting in various forms

## ALE Interfaces (1)

- Role of the ALE interface within the EPCglobal Network Architecture
  - Provide independence between the infrastructure components that acquire the raw EPC data
  - Architectural component(s) that filter & count that data, and the applications that use the data

## ALE Interfaces (2)

- Specify, in a high-level, declarative way, what data they are interested in or what operations they want performed, without dictating an implementation
- Standardized format for reporting accumulated, filtered data and results from carrying out operations that is largely independent of where the data originated or how it was processed

# ALE Interfaces (3)

- Abstracts the channels through which data carriers are accessed
  - Higher-level notion of “logical reader” (“location”)
  - Hide the details of what physical devices were used to interact with data relevant to a particular logical location

# ALE Interfaces (4)

- Abstracts the addressing of information stored on Tags and other data carriers
  - "fields"
  - Hide from clients the details of how a particular data element is encoded into a bit-level representation and stored at a particular address within a data carrier's memory
- Security mechanism
  - administrators may choose which operations a given application may perform

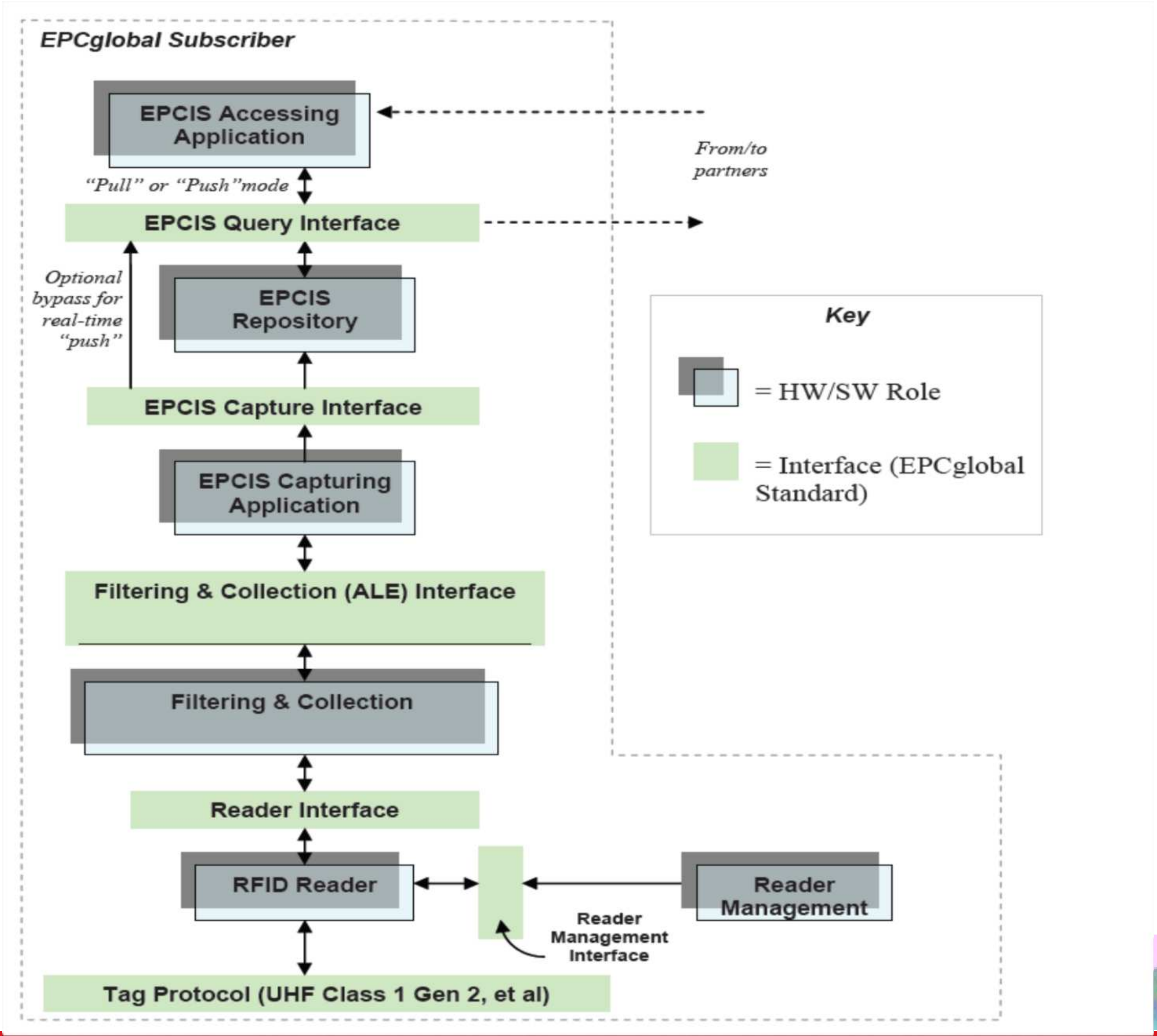
## Objectives of ALE and Difference from EPCIS (1)

- Drive as much filtering, counting, and other low-level processing as low in the architecture as possible
- Minimize the amount of “business logic” embedded in the tags, readers, embedded software/middleware
- ALE interface exclusively oriented towards real-time processing of EPC data

## Objectives of ALE and Difference from EPCIS (2)

- No persistent storage
- Events are pure statements of “what, where, and when”
- No business semantics expressed

# ALE in EPCglobal Network Architecture


EPCglobal Subscriber
EPCIS Accessing Application
"Pull" or "Push" mode
EPCIS Query Interface
From/to partners
Optional bypass for real-time "push"
EPCIS Repository
EPCIS Capture Interface
EPCIS Capturing Application
Filtering & Collection (ALE) Interface
Filtering & Collection
Reader Interface
RFID Reader
Reader Management
Reader Management Interface
Tag Protocol (UHF Class 1 Gen 2, et al)
Key
= HW/SW Role
= Interface (EPCglobal Standard)

## EPCglobal Network Architecture (1)

- Readers
  - – Multiple observations of RFID tags while they are in the read zone
- Reader Interface
  - – Control and delivery of raw tag reads from Readers to the Filtering & Collection role
  - – Events at this interface say “Reader A saw EPC X at time T”

## EPCglobal Network Architecture (2)

- Filtering & Collection
  - Filters and collects raw tag reads, over time intervals delimited by events defined by the EPCIS Capturing Application (e.g. tripping a motion detector)

## EPCglobal Network Architecture (3)

- Filtering & Collection (ALE) Interface
  - Control and delivery of filtered and collected tag read data from Filtering & Collection role to the EPCIS Capturing Application role. Events at this interface say “At Location L, between time T1 and T2, the following EPCs were observed,” where the list of EPCs has no duplicates and has been filtered by criteria defined by the EPCIS Capturing Application.

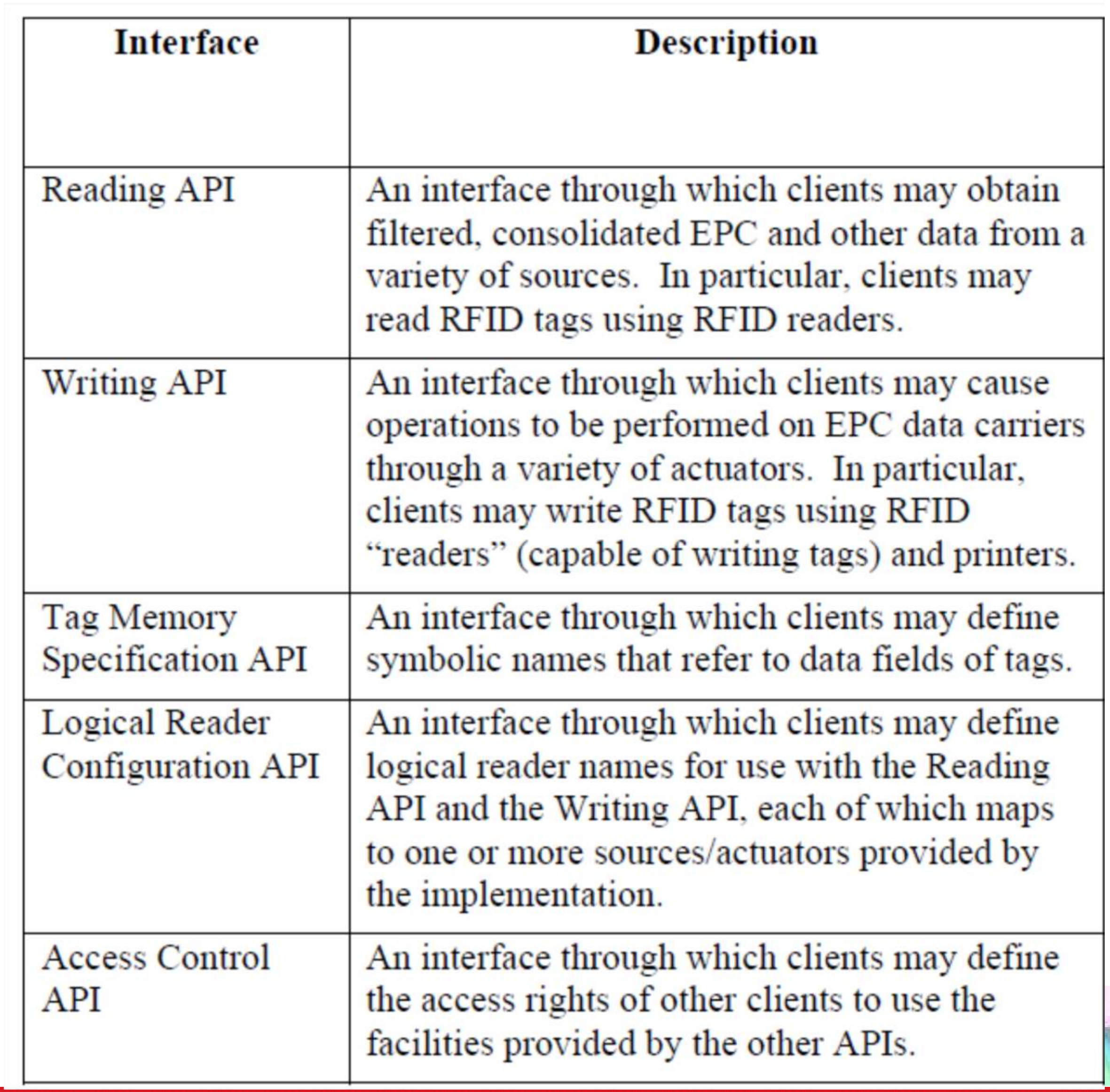
## EPCglobal Network Architecture (4)

- EPCIS Capturing Application
  - Supervises the operation of the lower EPC elements, and provides business context by coordinating with other sources of information involved in executing a particular step of a business process.
- EPCIS Capture Interface
  - EPCIS data is delivered to enterprise-level roles, including EPCIS Repositories, EPCIS Accessing Applications, and data exchange with partners

## EPCglobal Network Architecture (5)

- EPCIS Accessing Application
  - Carrying out overall enterprise business processes, such as warehouse management, shipping and receiving, historical throughput analysis, and so forth, aided by EPC-related data.
- EPCIS Repository Records
  - EPCIS-level events generated by one or more EPCIS Capturing Applications, and makes them available for later query by EPCIS Accessing Applications

## ALE Interfaces



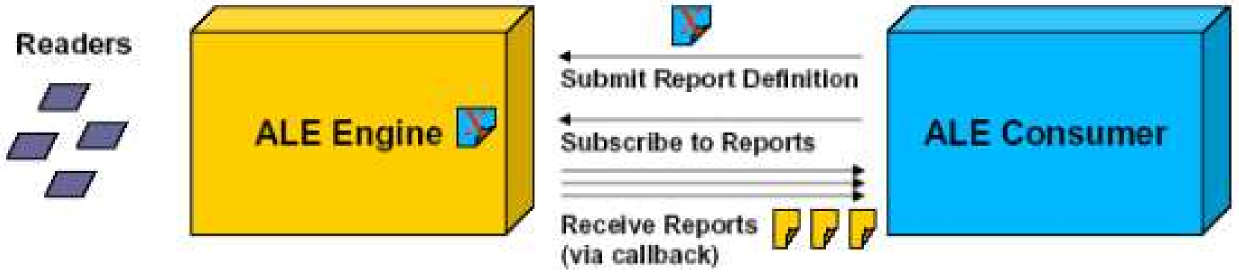
| Interface | Description |
|---|---|
| Reading API | An interface through which clients may obtain filtered, consolidated EPC and other data from a variety of sources. In particular, clients may read RFID tags using RFID readers. |
| Writing API | An interface through which clients may cause operations to be performed on EPC data carriers through a variety of actuators. In particular, clients may write RFID tags using RFID "readers" (capable of writing tags) and printers. |
| Tag Memory Specification API | An interface through which clients may define symbolic names that refer to data fields of tags. |
| Logical Reader Configuration API | An interface through which clients may define logical reader names for use with the Reading API and the Writing API, each of which maps to one or more sources/actuators provided by the implementation. |
| Access Control API | An interface through which clients may define the access rights of other clients to use the facilities provided by the other APIs. |

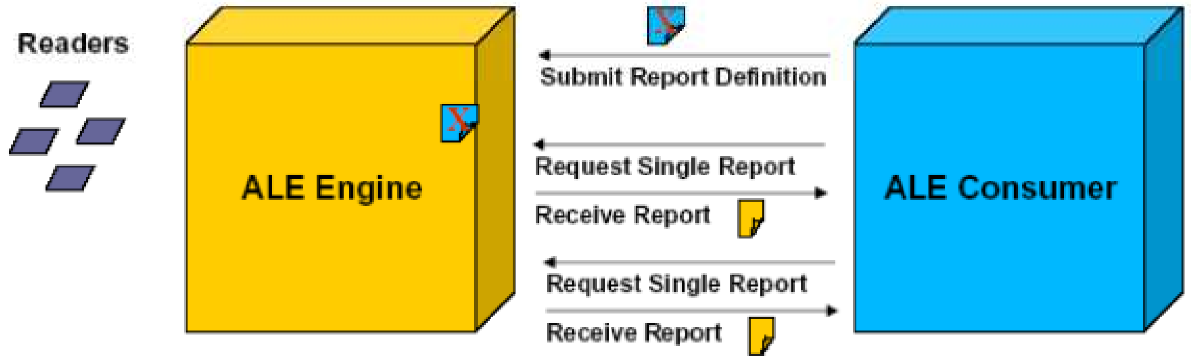
## API Interaction

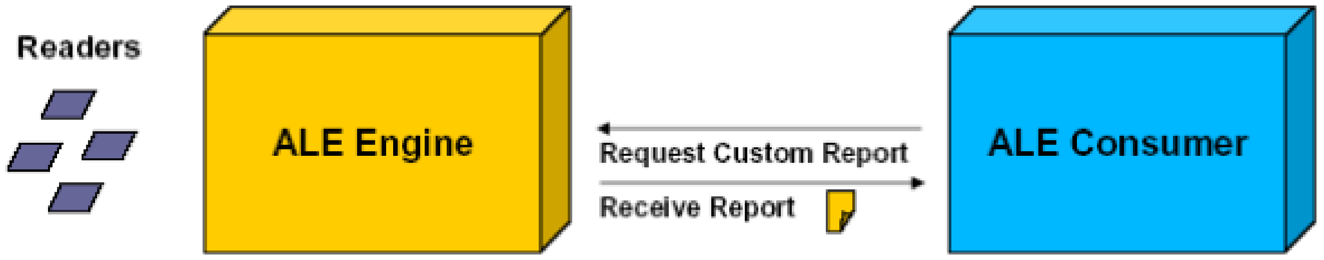
- General interaction model
  - One or more clients that make method calls to an interface class corresponding to an API
- Reading API and Writing API
  - Provides a way for clients to subscribe to events that are delivered asynchronously

# ALE operation modes

## poll, immediate, and subscribe…







# Fundamental ALE Concepts (1)

- Purpose of the ALE interface
  - Allow business applications to read and operate upon tags
  - ALE was primarily conceived and developed in the context of RFID tags, the interface is designed to be general enough to accommodate other kinds of data carriers, such as bar codes, OCR text, and in some instances even human interaction through a keyboard or display

## Fundamental ALE Concepts (2)

- Reader cycle
  - Smallest unit of interaction with a Reader
- The "Reader" is the communication pathway between the ALE subsystem and the RF protocol subsystem
  - A reader cycle might represent one iteration of the RF protocol used to communicate with RFID tags

# Fundamental ALE Concepts (3)

- Event cycle or command cycle
  - Interval of time over which an ALE implementation carries out interactions with one or more Readers on behalf of an ALE client
- Report
  - Response sent from the ALE implementation to the ALE client at the conclusion of an event cycle or command cycle

## Fundamental ALE Concepts (4)

- During an event cycle or command cycle
  - An ALE implementation carries out one or more reader cycles with the designated Readers
  - Through those reader cycles carry out the wishes of the ALE client for that event cycle or command cycle

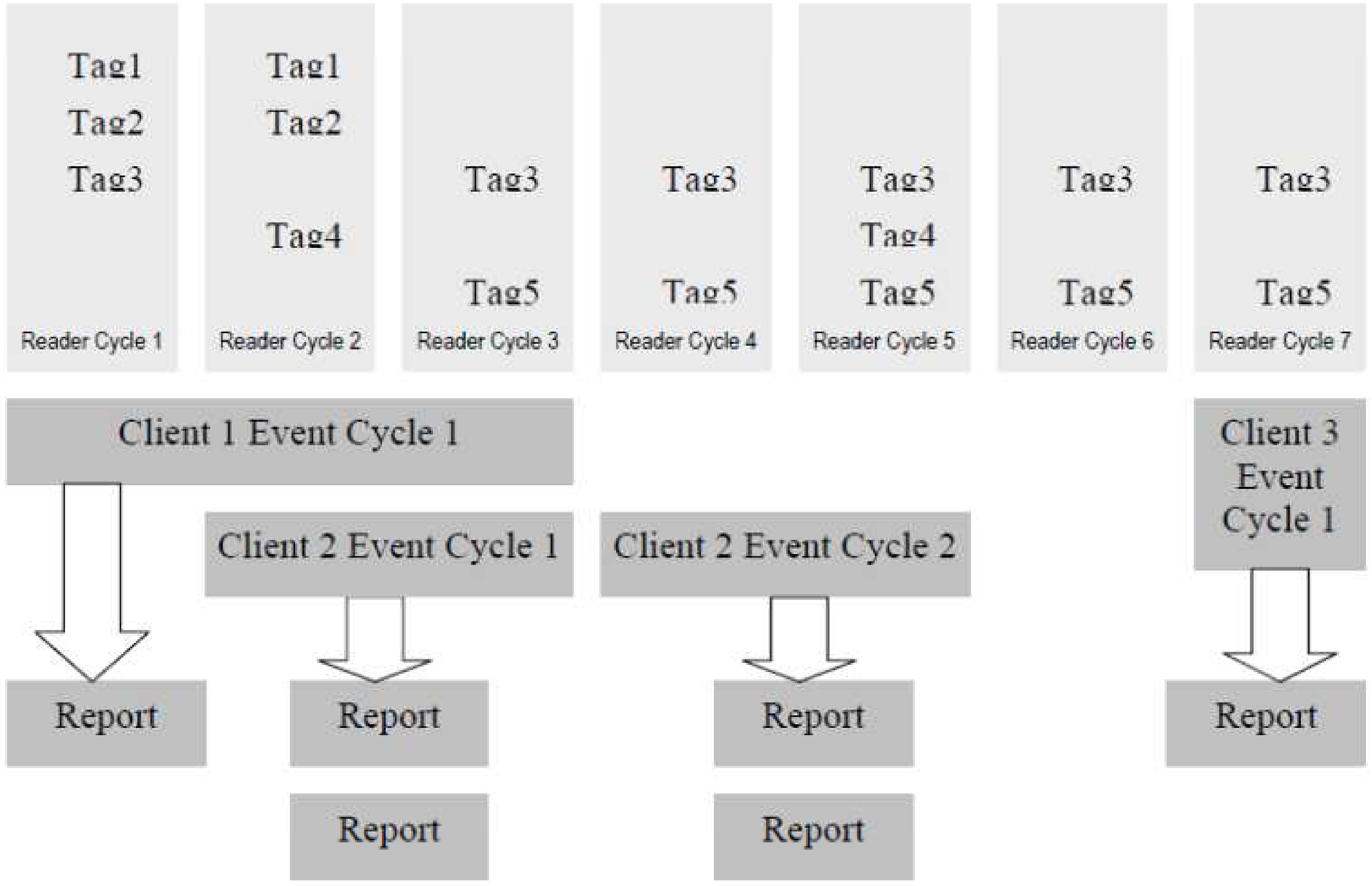
## Event Cycles (1)

- Event cycle
  - Smallest unit of interaction between an ALE client and an ALE implementation through the ALE Reading API
  - Interval of time during which Tags are read
  - At the conclusion of an event cycle, a report is sent to the ALE client containing information read from the Tags
  - As Tags move in and out of the detection zone of a Reader, tag data reported changes
  - Within an event cycle, the same Tag may be read several times

## Event Cycles (2)

- An ALE client may specify that an event cycle may:
  - Extend for a specified duration
  - Occur periodically
  - Be triggered by external events
  - Be delimited when no new Tags are detected by any Reader specified for that event cycle for a specified interval of time
  - Terminate when any Reader specified for that event cycle reports a new Tag to the ALE implementation, thus delivering data to the ALE client as soon as it is known to the ALE implementation

# Event Cycles (3)



# Event Cycles (4)

- ALE Clients get information about event cycles through reports
  - What set *R* to report
  - An optional filter *F(R)* to apply
  - Whether to report

## What to report? (1)

- The complete set from the current event cycle *R* = *Ecur*, or
- The differential set that only includes differences of the current event cycle relative to the previous one (assuming the same event cycle boundaries). This can be the set of additions *R* = *(Ecur – Eprev)* or the set of deletions *R* = *(Eprev – Ecur)*, where '–' denotes the set difference operator

## What to report? (2)

- Filtering
  - Includes some Tags and excludes others based on the data contained in their fields

## Whether to report

- The members of the set, *F(R)* (*i.e.*, the tag data themselves). In this case, the ALE client also specifies which data fields to report for each Tag, and how the data is to be formatted for consumption by the client
- The quantity, or cardinality, of the set |*F(R)*|, or of the groups making up the set

## ALE Layer API interaction

- Event cycle specification (ECSpec), which specifies
  - One or more Readers
  - Event cycle boundaries
  - Set of reports as defined above
- ALE Layer
  - Responds by returning the information implied by that report specification for one or more event cycles

## Group Reports (1)

- Sometimes it is useful to group Tags read during an event cycle based on portions of the EPC or other fields
- Grouping Operator
  - Function that maps tag data into some sort of group cod
  - E.g.,: A grouping operator might map the EPC field of a tag into a GTIN group, or simply into the upper bits (manufacturer and product) of the EPC

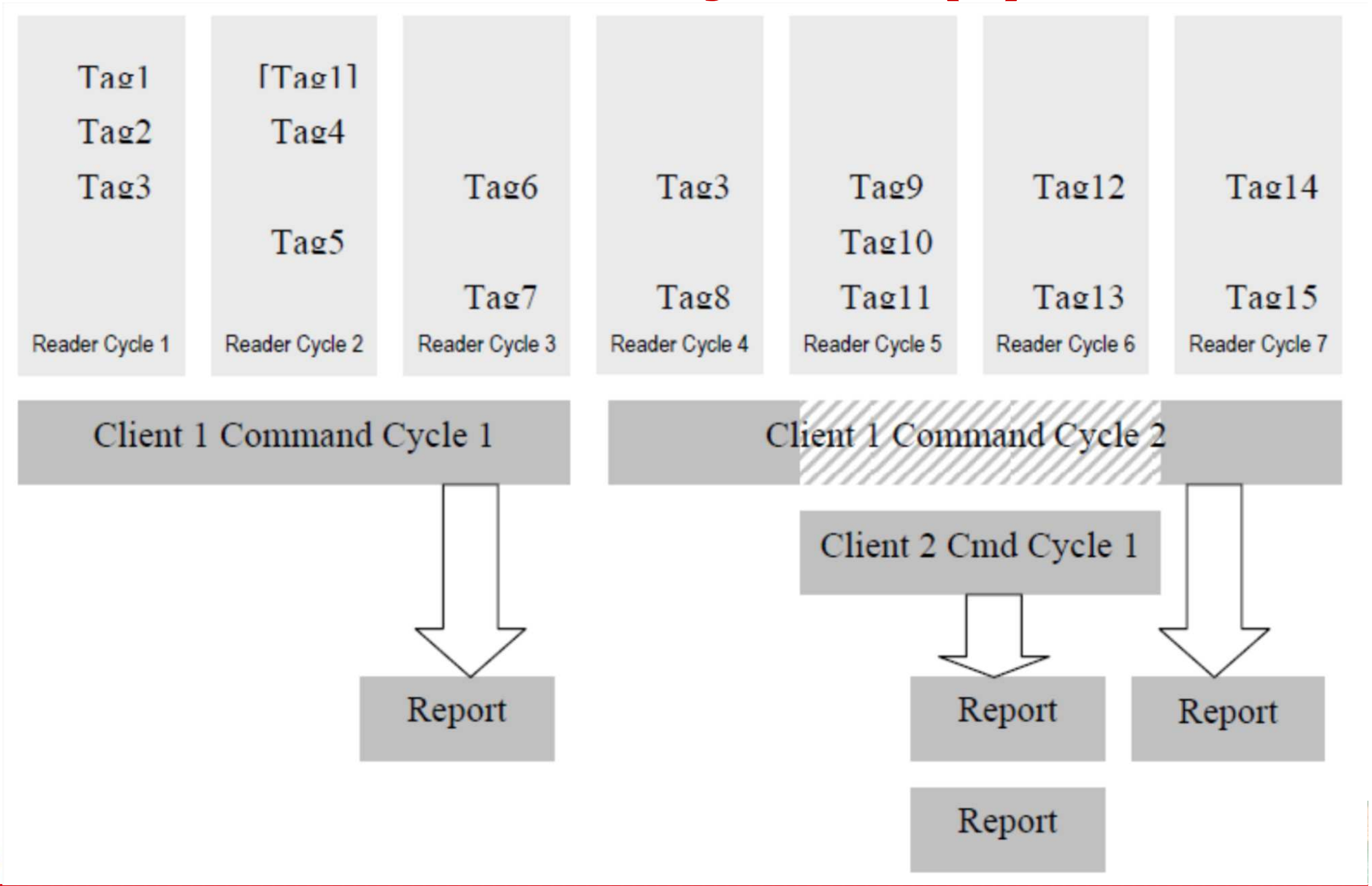
## Group Reports (2)

- Group membership report
  - Set of pairs, where the first element in each pair is a group name
  - Second element is the list of EPCs that fall into that group, i.e., S↓g
- Group cardinality report
  - Instead of enumerating the EPCs in each group, the group cardinality report just reports how many of each there are

## Command Cycles (1)

- Command cycle
  - Smallest unit of interaction between an ALE client and an ALE implementation through the ALE Writing API
  - An interval of time during which Tags are written, or other operations performed upon them

# Command Cycles (2)



## Interaction between an ALE client and an ALE implementation through the Writing API (1)

- A client provides to the ALE implementation a command cycle specification 1049 (CCSpec), which specifies
  - one or more Readers (this is done indirectly)
  - command cycle boundaries, and
  - a set of command lists to apply to Tags. Each command list includes
    - a filter that specifies which Tags to operate upon, and
    - an ordered list of operations to perform on each Tag that matches the filter.

## Interaction between an ALE client and an ALE implementation through the Writing API (2)

- The ALE Layer responds by carrying out the operations on Tags, and returning a 1056 report that describes what Tags were encountered and what processing was performed upon them.

## Difference between event cycles and command cycles (1)

- Event cycles only read Tags, without changing their contents or performing other side-effects upon them
- It is possible for several simultaneously active event cycles to share the result of a single reader cycle, and an ALE implementation MAY share reader cycles in this way
- Because simultaneous event cycles may have different boundaries, it MAY be necessary for the ALE implementation to read a given Tag more than once

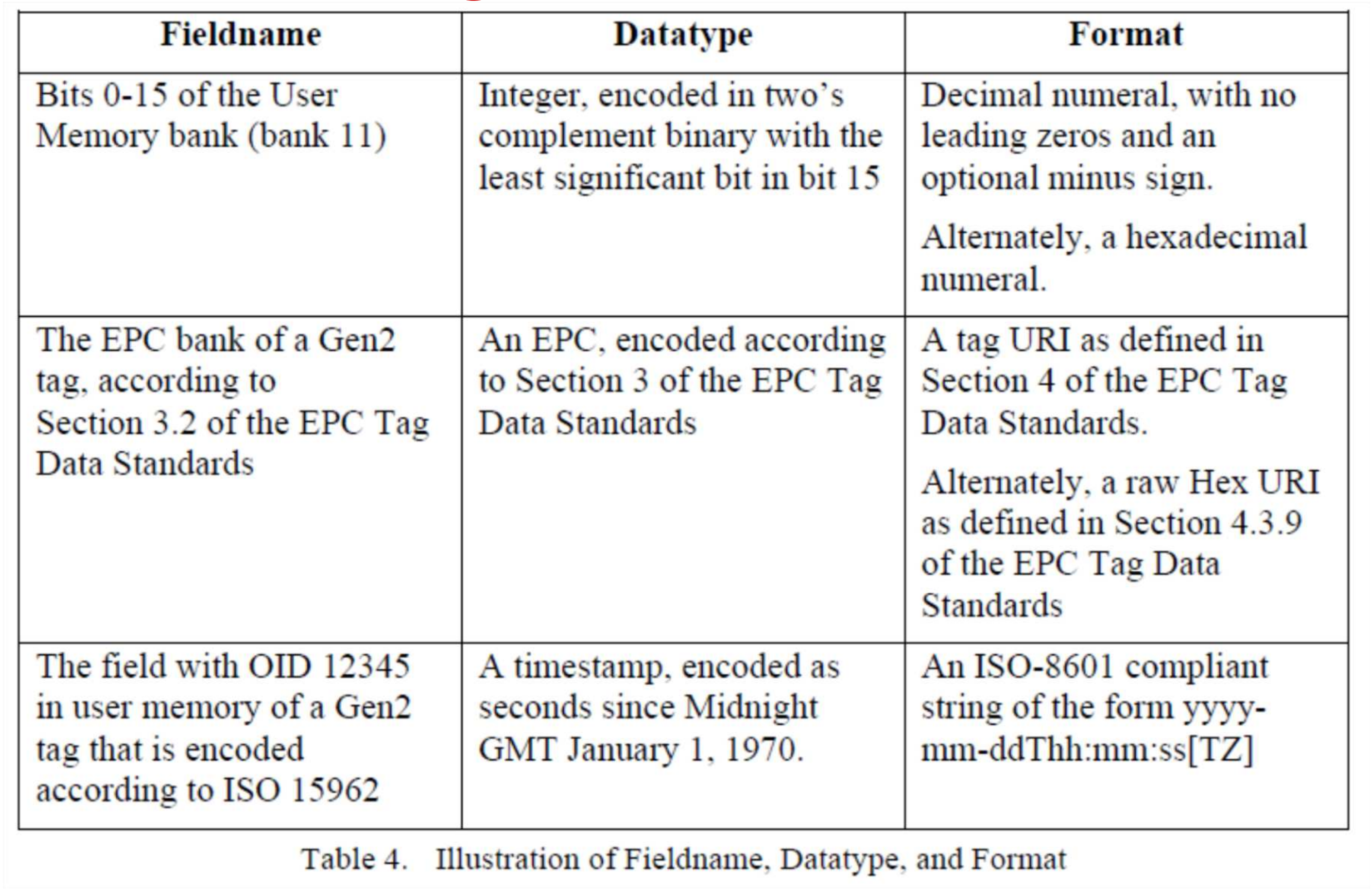
## Difference between event cycles and command cycles (2)

- Command cycles may write Tags and perform other side-effects such as killing or locking
- Simultaneous command cycles are permitted in the ALE Writing API, but it is not expected that reader cycles will be shared. This is both because simultaneous command cycles are likely to be operating upon disjoint sets of Tags or performing disjoint operations on them, and because each command cycle may need to do its own bookkeeping to avoid duplicates

## Tag Data Model

- A fieldname
  - Specifies which data field of the Tag to operate upon
- The datatype
  - Specifies what kind of data values that field is considered to contain, and how they are encoded into the Tag memory
- A format
  - Specifies the syntax by which individual data values are presented at the level of the ALE API (that is, the format of data values as reported by the ALE API when fields are read, and the format of data values provided by the ALE client

## Tag Data Model



| Fieldname | Datatype | Format |
|---|---|---|
| Bits 0-15 of the User Memory bank (bank 11) | Integer, encoded in two's complement binary with the least significant bit in bit 15 | Decimal numeral, with no leading zeros and an optional minus sign.<br><br>Alternately, a hexadecimal numeral. |
| The EPC bank of a Gen2 tag, according to Section 3.2 of the EPC Tag Data Standards | An EPC, encoded according to Section 3 of the EPC Tag Data Standards | A tag URI as defined in Section 4 of the EPC Tag Data Standards.<br><br>Alternately, a raw Hex URI as defined in Section 4.3.9 of the EPC Tag Data Standards |
| The field with OID 12345 in user memory of a Gen2 tag that is encoded according to ISO 15962 | A timestamp, encoded as seconds since Midnight GMT January 1, 1970. | An ISO-8601 compliant string of the form yyyy-mm-ddThh:mm:ss[TZ] |

Table 4. Illustration of Fieldname, Datatype, and Format

## Kinds of Fieldnames (1)

- Fixed-address fieldnames of the form *@bank.length[.offset]*, where *bank*, *length*, and *offset* are integers
- A fieldname of this form specifies a fixed field comprising *length* contiguous bits, starting at fixed bit location *offset* within bank *bank* of tag memory

# Kinds of Fieldnames (2)

- Variable fieldnames of the form *@bank.oid*, where *bank* is an integer and *oid* is 1117 an object identifier expressed as a URN according to [RFC3061]
- A fieldname of this form specifies a variable field encoded according to ISO 15962 [ISO15962].

## Kinds of Fieldnames (3)

- A symbolic fieldname that is a user- or implementation-defined string, not beginning 1121 with an atsign (@) character.

## “Field Not Found” Condition

- When an ALE implementation accesses a particular Tag during an event cycle or command cycle, it may be that the Tag does not have a field that is specified in the governing ECSpec or CCSpec

## Behavior in the Reading API (1)

- If the field was included in the *primaryKeyFields* list, it causes the Tag to be omitted from the event cycle
- If the field was included in an *ECFilterSpec*, it causes the Tag to be omitted from the event cycle
- If the field was included in an *ECGroupSpec*, it causes the Tag to be assigned to the default group

# Behavior in the Reading API (2)

- If the field was included in an ECReportOutputSpec, it causes the value to be reported as null
- If the field was included in an ECFilterSpec, it causes the Tag to be omitted from the command cycle
- If the field was included in a CCOpSpec, it causes the operation to be reported with a FIELD_NOT_FOUND_ERROR status code

## Reader Cycle Timing (1)

- Clients may specify the boundaries of event cycles and command cycles, which accumulate data from or manipulate tags during one or more underlying reader cycles, but the API does not provide a client with explicit control over the frequency at which reader cycles are completed

# Reader Cycle Timing (2)

- Clients may specify the boundaries of event cycles and command cycles, which accumulate data from or manipulate tags during one or more underlying reader cycles, but the API does not provide a client with explicit control over the frequency at which reader cycles are completed

## Reader Cycle Timing (3)

- A client or clients may make simultaneous requests for event cycles that may have differing event cycle boundaries and different report specifications. In this case, clients must necessarily share a common view of when and how frequently reader cycles take place

## Reader Cycle Timing (4)

- In cases where there are many RFID readers in physical proximity (perhaps communicating to different ALE implementations), the reader cycle frequency must be carefully tuned and coordinated to avoid reader interference

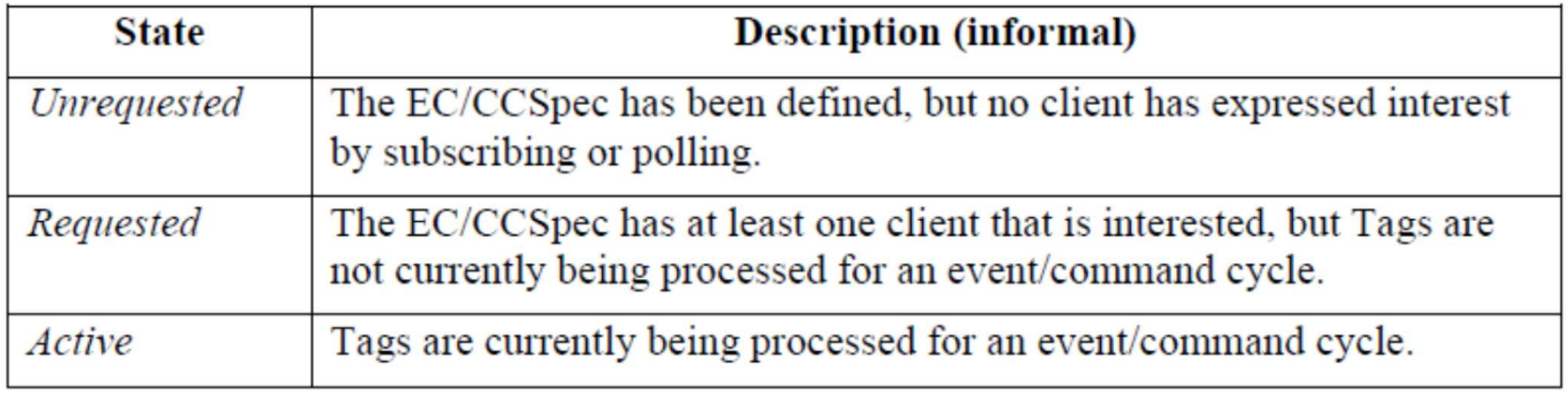
## Execution of Event Cycles and Command Cycles (1)

- An event cycle specification (ECSpecs) or a command cycle specification (CCSpecs) comes into existence through a client interacting with the ALE Reading API or the ALE Writing API, respectively

## Execution of Event Cycles and Command Cycles (2)

- A standing EC/CCSpec may be posted using the define method of the Reading/Writing API
- One or more clients may subscribe to that EC/CCSpec using the subscribe method
- The EC/CCSpec will execute event/command cycles as long as there is at least one subscriber. A poll call is like subscribing then unsubscribing immediately after one event/command cycle is completed

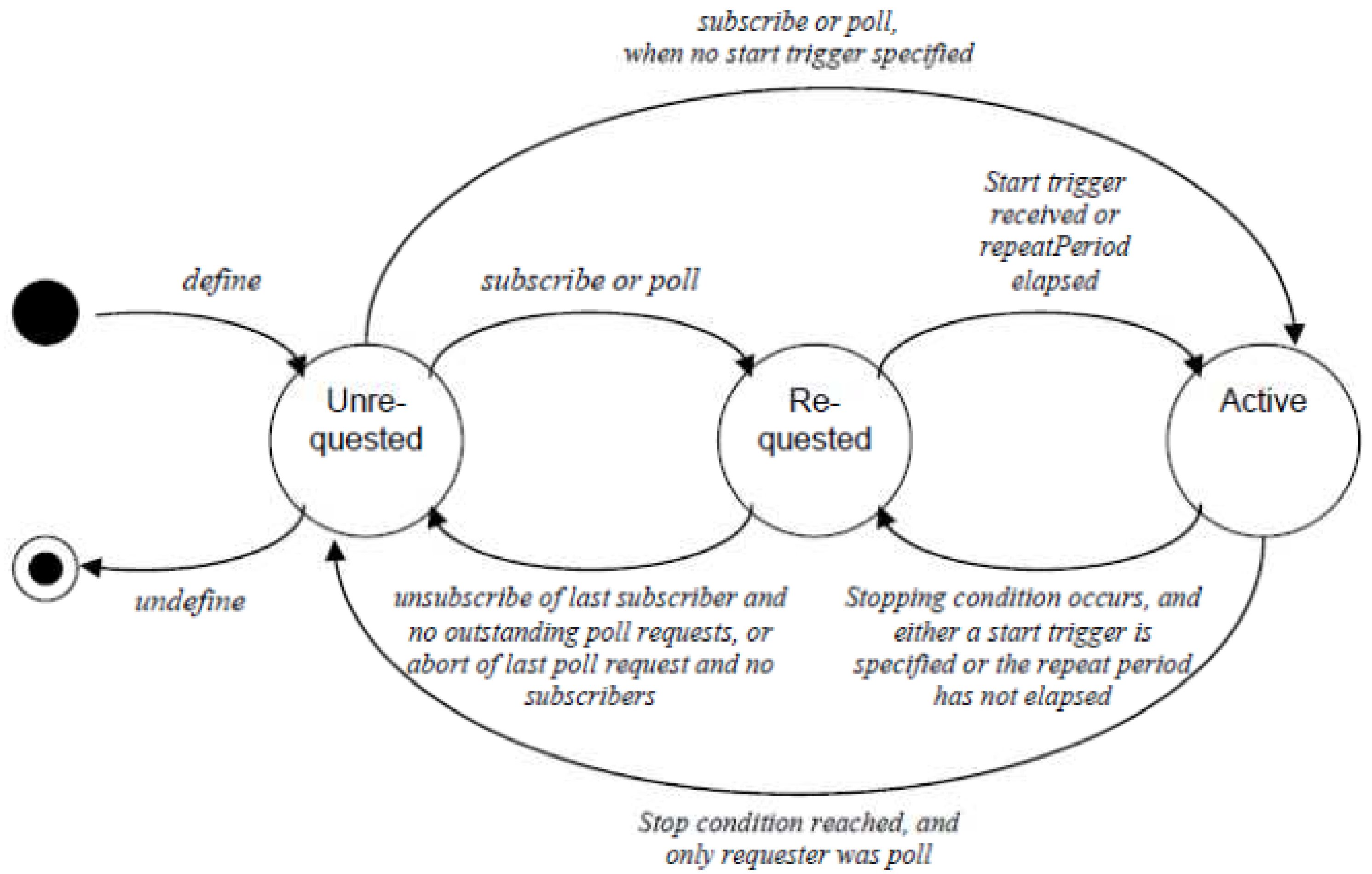
## EC/CCSpecs Lifecycle



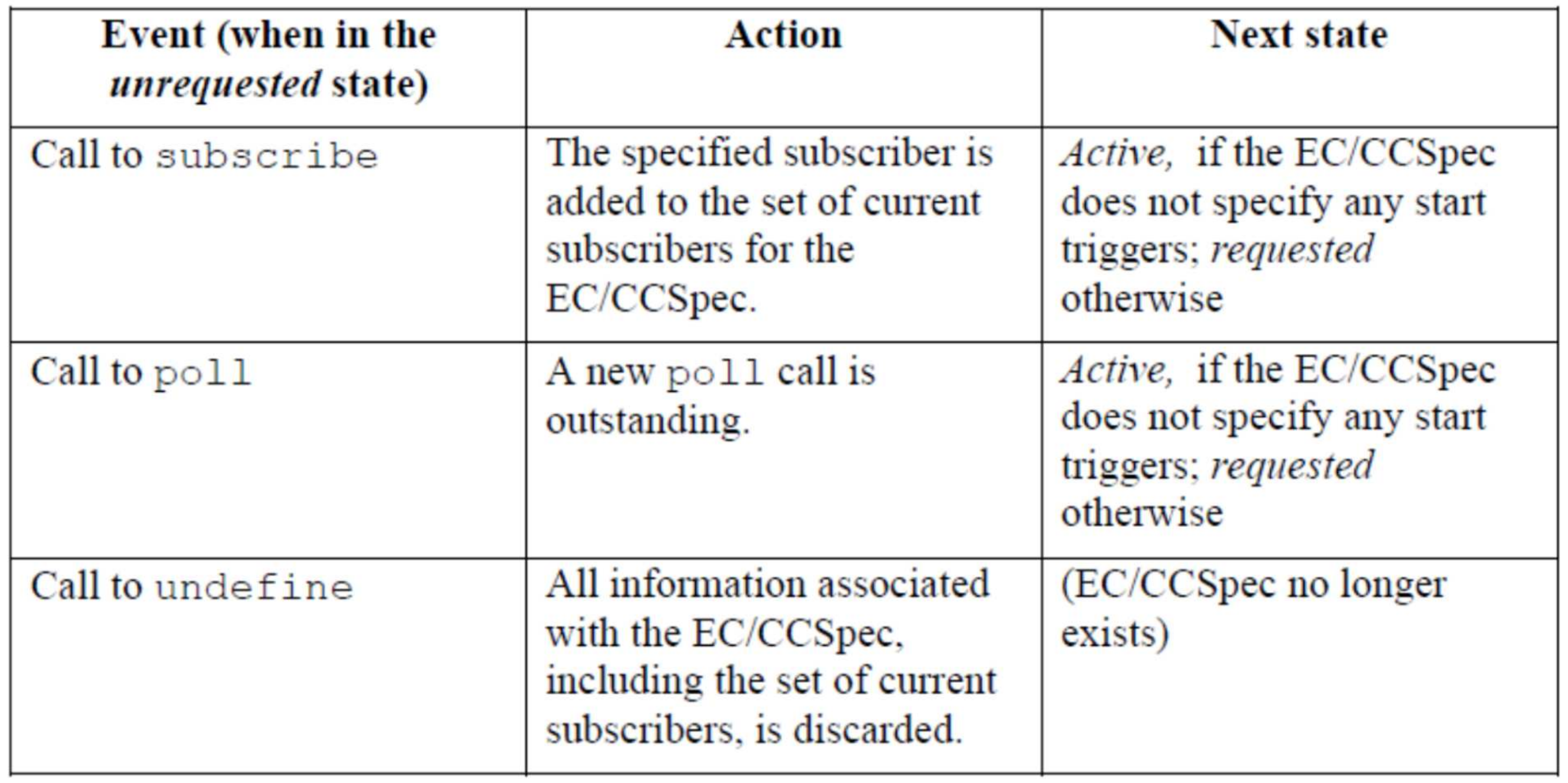
| State | Description (informal) |
|---|---|
| *Unrequested* | The EC/CCSpec has been defined, but no client has expressed interest by subscribing or polling. |
| *Requested* | The EC/CCSpec has at least one client that is interested, but Tags are not currently being processed for an event/command cycle. |
| *Active* | Tags are currently being processed for an event/command cycle. |

Table 5. EC/CCSpec Lifecycle States

## Lifecycle State Transitions for EC/CCSpecs (1)


subscribe or poll,
when no start trigger specified
Start trigger
received or
repeatPeriod
elapsed
define
subscribe or poll
Unre-
quested
Re-
quested
Active
undefine
unsubscribe of last subscriber and
no outstanding poll requests, or
abort of last poll request and no
subscribers
Stopping condition occurs, and
either a start trigger is
specified or the repeat period
has not elapsed
Stop condition reached, and
only requester was poll

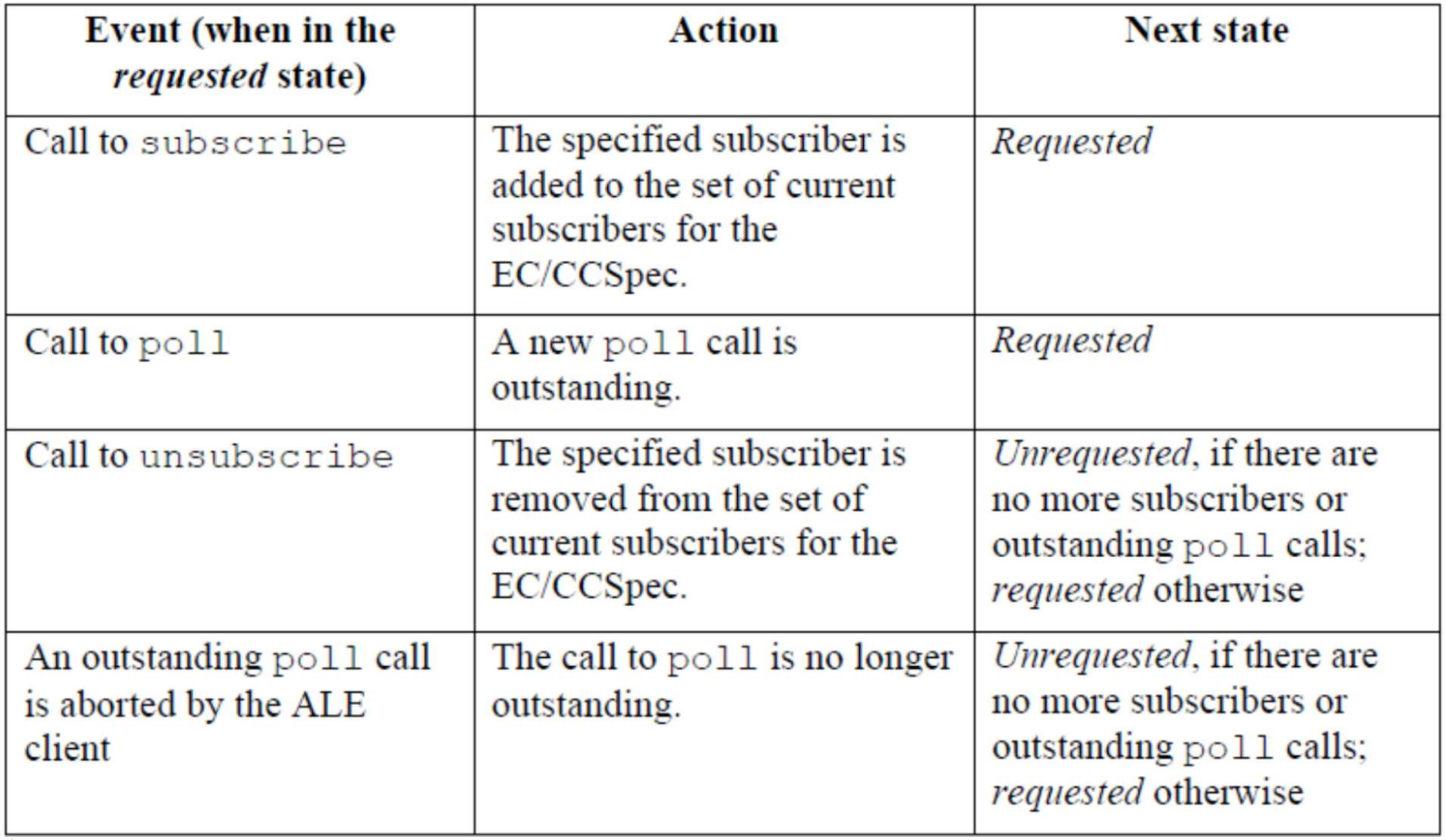
## Lifecycle State Transitions for EC/CCSpecs (2)



| Event (when in the *unrequested* state) | Action | Next state |
|---|---|---|
| Call to `subscribe` | The specified subscriber is added to the set of current subscribers for the EC/CCSpec. | *Active*, if the EC/CCSpec does not specify any start triggers; *requested* otherwise |
| Call to `poll` | A new `poll` call is outstanding. | *Active*, if the EC/CCSpec does not specify any start triggers; *requested* otherwise |
| Call to `undefine` | All information associated with the EC/CCSpec, including the set of current subscribers, is discarded. | (EC/CCSpec no longer exists) |

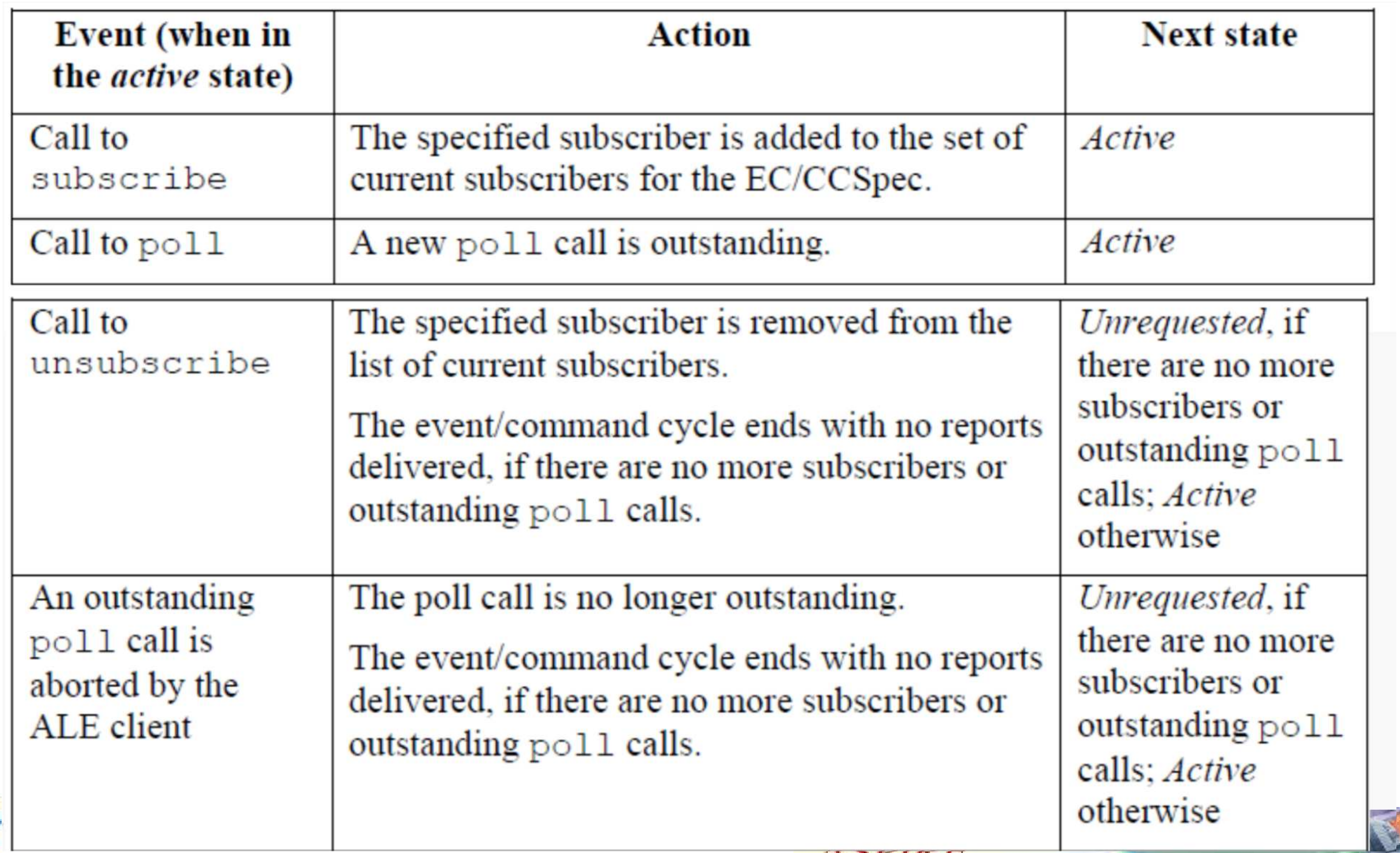
Table 6. State Transitions from the Unrequested State

## Lifecycle State Transitions for EC/CCSpecs (3)



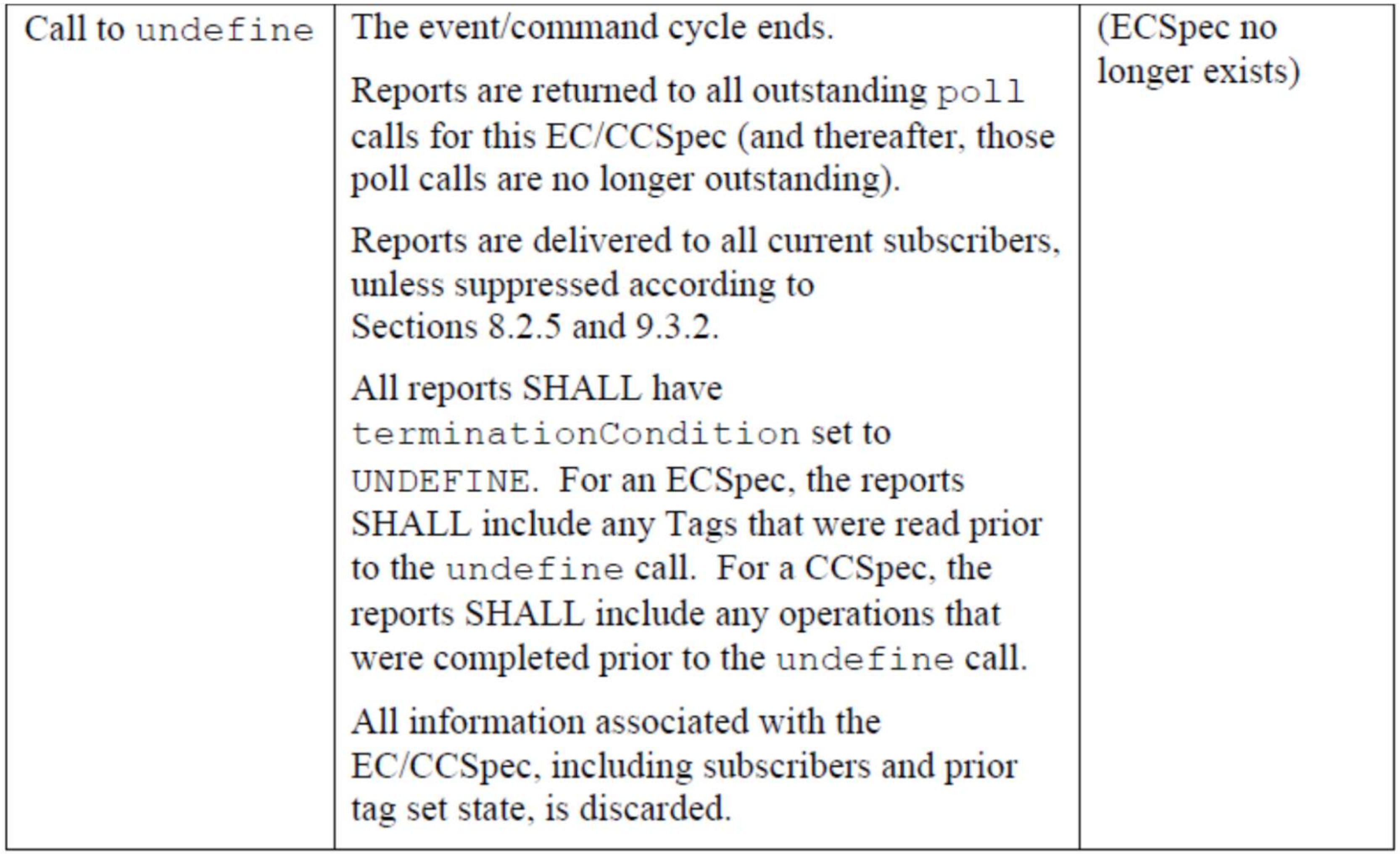
| **Event (when in the *requested* state)** | **Action** | **Next state** |
|---|---|---|
| Call to `subscribe` | The specified subscriber is added to the set of current subscribers for the EC/CCSpec. | *Requested* |
| Call to `poll` | A new `poll` call is outstanding. | *Requested* |
| Call to `unsubscribe` | The specified subscriber is removed from the set of current subscribers for the EC/CCSpec. | *Unrequested*, if there are no more subscribers or outstanding `poll` calls; *requested* otherwise |
| An outstanding `poll` call is aborted by the ALE client | The call to `poll` is no longer outstanding. | *Unrequested*, if there are no more subscribers or outstanding `poll` calls; *requested* otherwise |

## Lifecycle State Transitions for EC/CCSpecs (4)



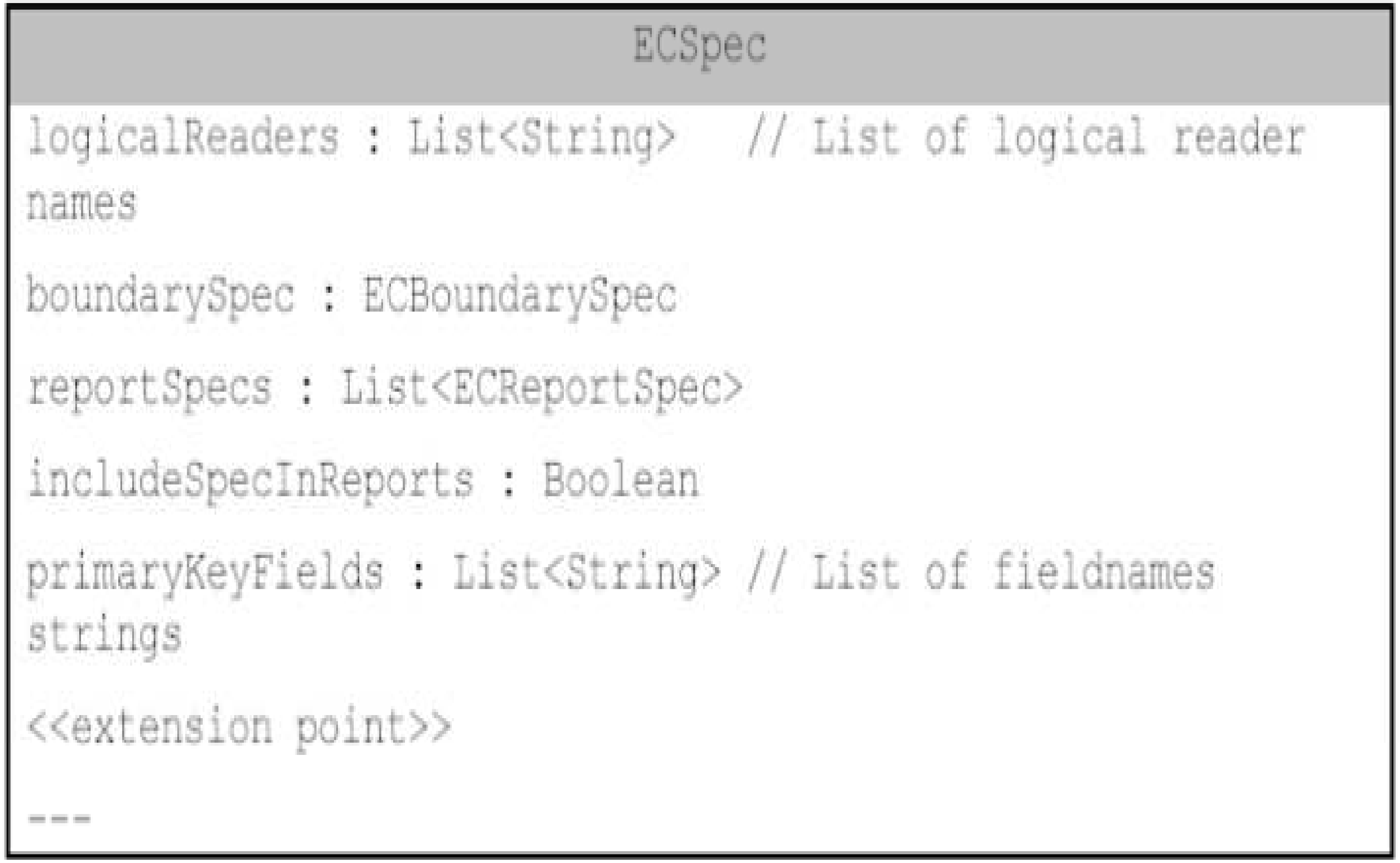
| Event (when in the *active* state) | Action | Next state |
|---|---|---|
| Call to `subscribe` | The specified subscriber is added to the set of current subscribers for the EC/CCSpec. | *Active* |
| Call to `poll` | A new `poll` call is outstanding. | *Active* |
| Call to `unsubscribe` | The specified subscriber is removed from the list of current subscribers.<br><br>The event/command cycle ends with no reports delivered, if there are no more subscribers or outstanding `poll` calls. | *Unrequested*, if there are no more subscribers or outstanding `poll` calls; *Active* otherwise |
| An outstanding `poll` call is aborted by the ALE client | The poll call is no longer outstanding.<br><br>The event/command cycle ends with no reports delivered, if there are no more subscribers or outstanding `poll` calls. | *Unrequested*, if there are no more subscribers or outstanding `poll` calls; *Active* otherwise |

## Lifecycle State Transitions for EC/CCSpecs (5)



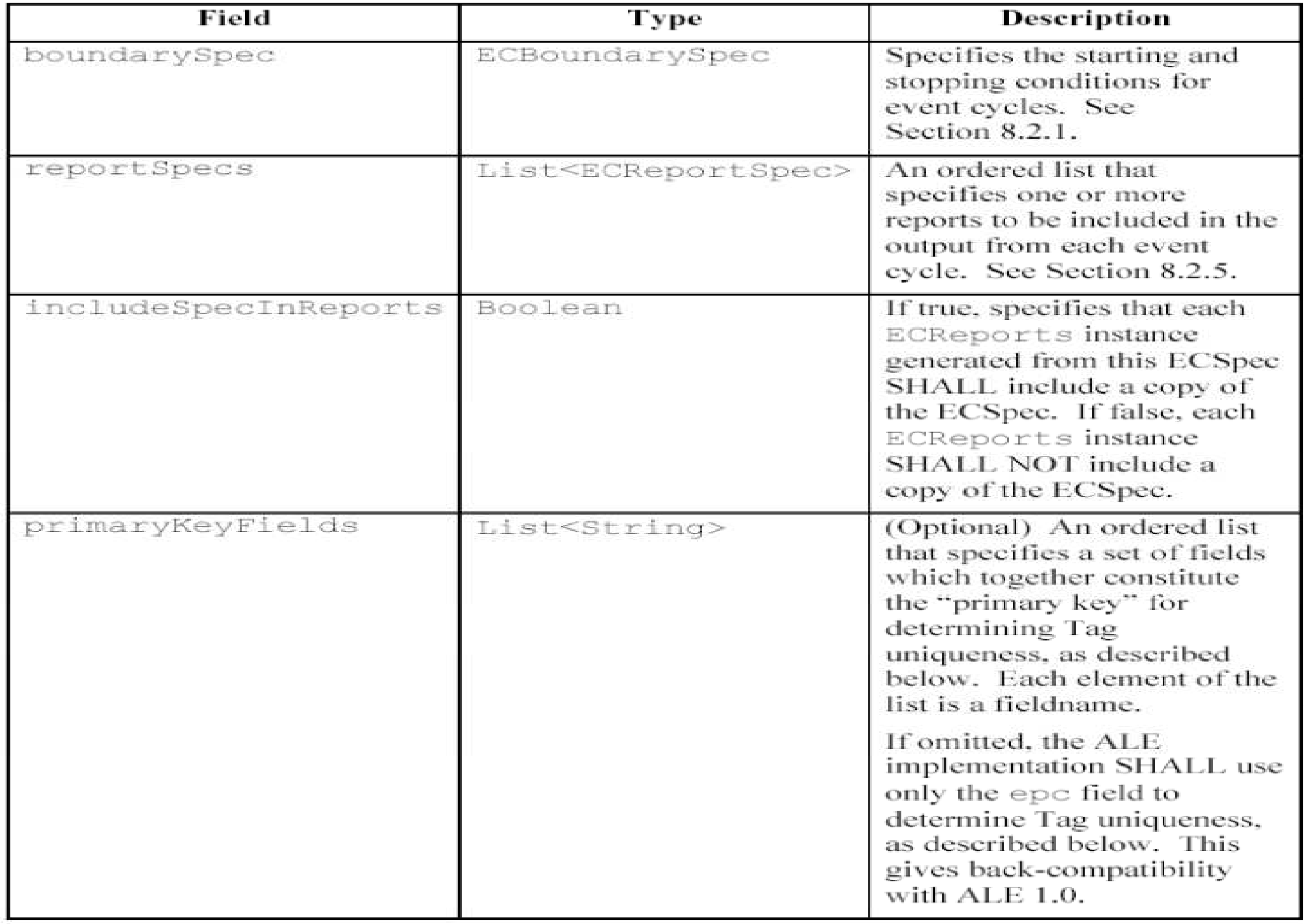
| | | |
|---|---|---|
| Call to `undefine` | The event/command cycle ends.<br><br>Reports are returned to all outstanding `poll` calls for this EC/CCSpec (and thereafter, those poll calls are no longer outstanding).<br><br>Reports are delivered to all current subscribers, unless suppressed according to Sections 8.2.5 and 9.3.2.<br><br>All reports SHALL have `terminationCondition` set to `UNDEFINE`. For an ECSpec, the reports SHALL include any Tags that were read prior to the `undefine` call. For a CCSpec, the reports SHALL include any operations that were completed prior to the `undefine` call.<br><br>All information associated with the EC/CCSpec, including subscribers and prior tag set state, is discarded. | (ECSpec no longer exists) |

# ECSpec (1)



```
ECSpec
logicalReaders : List<String>   // List of logical reader
names
boundarySpec : ECBoundarySpec
reportSpecs : List<ECReportSpec>
includeSpecInReports : Boolean
primaryKeyFields : List<String> // List of fieldnames
strings
<<extension point>>
---
```

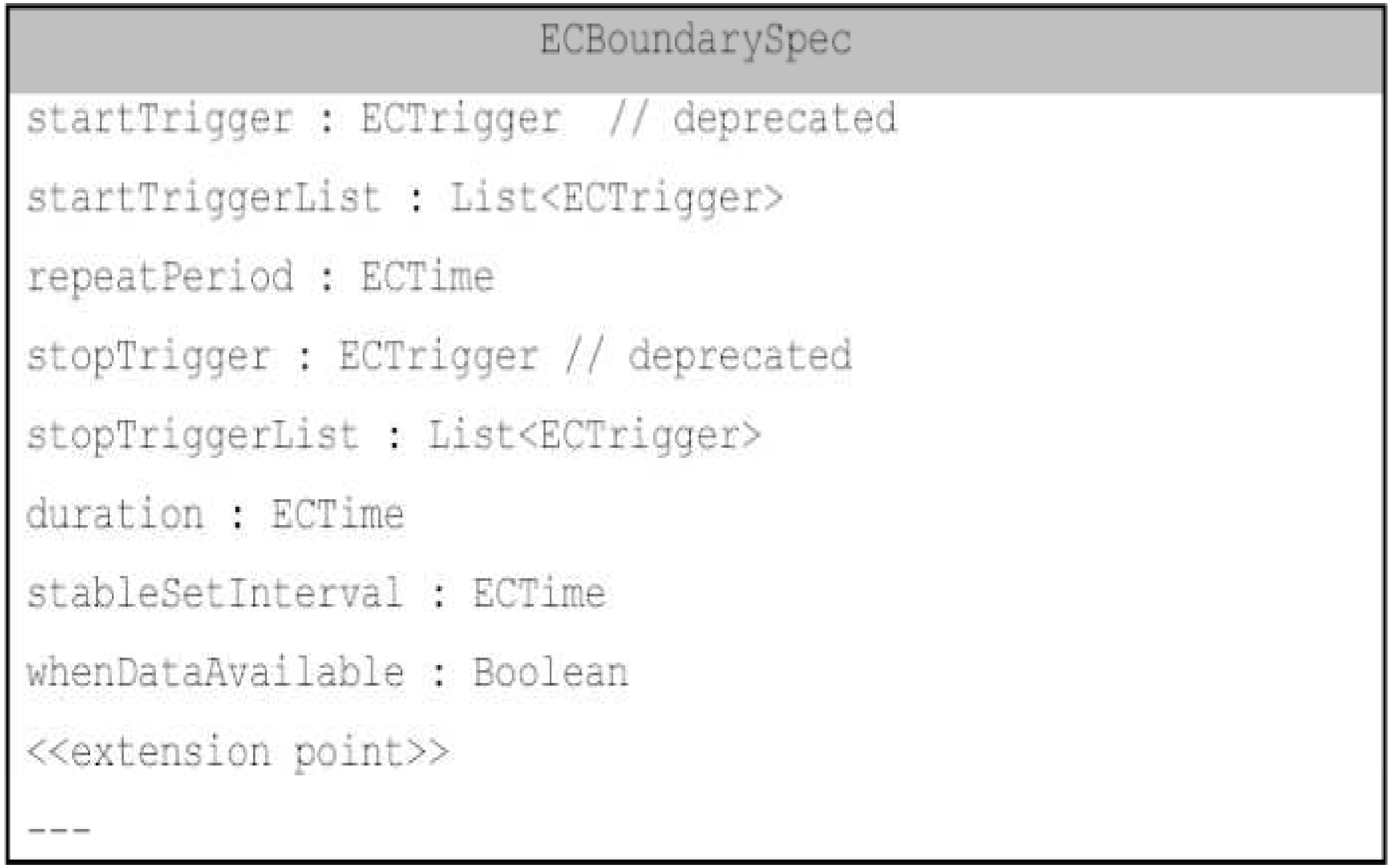
# ECSpec (2)



| Field | Type | Description |
|---|---|---|
| boundarySpec | ECBoundarySpec | Specifies the starting and stopping conditions for event cycles. See Section 8.2.1. |
| reportSpecs | List<ECReportSpec> | An ordered list that specifies one or more reports to be included in the output from each event cycle. See Section 8.2.5. |
| includeSpecInReports | Boolean | If true, specifies that each ECReports instance generated from this ECSpec SHALL include a copy of the ECSpec. If false, each ECReports instance SHALL NOT include a copy of the ECSpec. |
| primaryKeyFields | List<String> | (Optional) An ordered list that specifies a set of fields which together constitute the "primary key" for determining Tag uniqueness, as described below. Each element of the list is a fieldname.<br><br>If omitted, the ALE implementation SHALL use only the epc field to determine Tag uniqueness, as described below. This gives back-compatibility with ALE 1.0. |

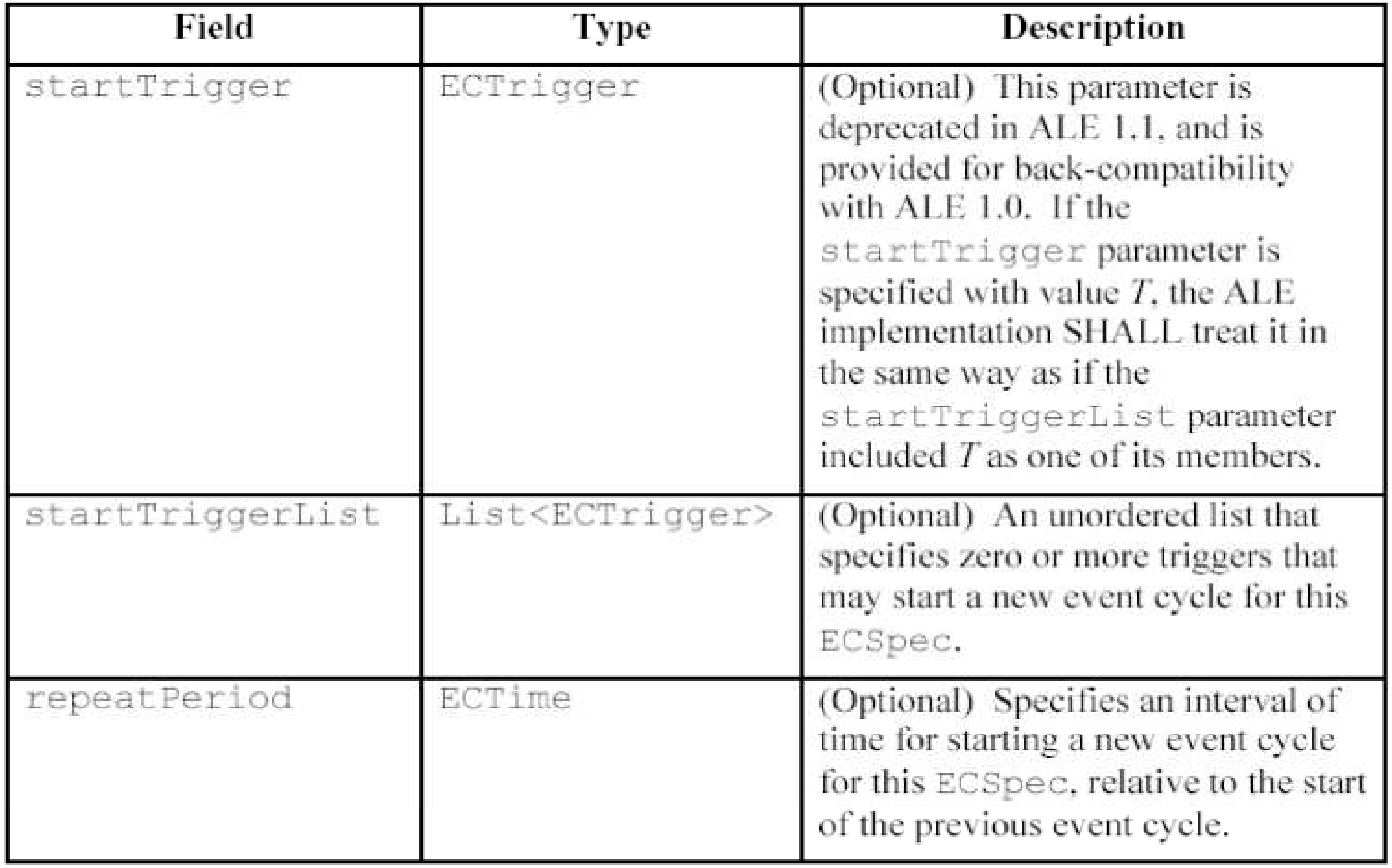
Table 31. ECSpec Fields

# ECBoundarySpec (1)

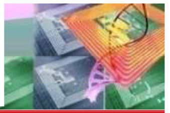


```
ECBoundarySpec
startTrigger : ECTrigger  // deprecated
startTriggerList : List<ECTrigger>
repeatPeriod : ECTime
stopTrigger : ECTrigger // deprecated
stopTriggerList : List<ECTrigger>
duration : ECTime
stableSetInterval : ECTime
whenDataAvailable : Boolean
<<extension point>>
---
```

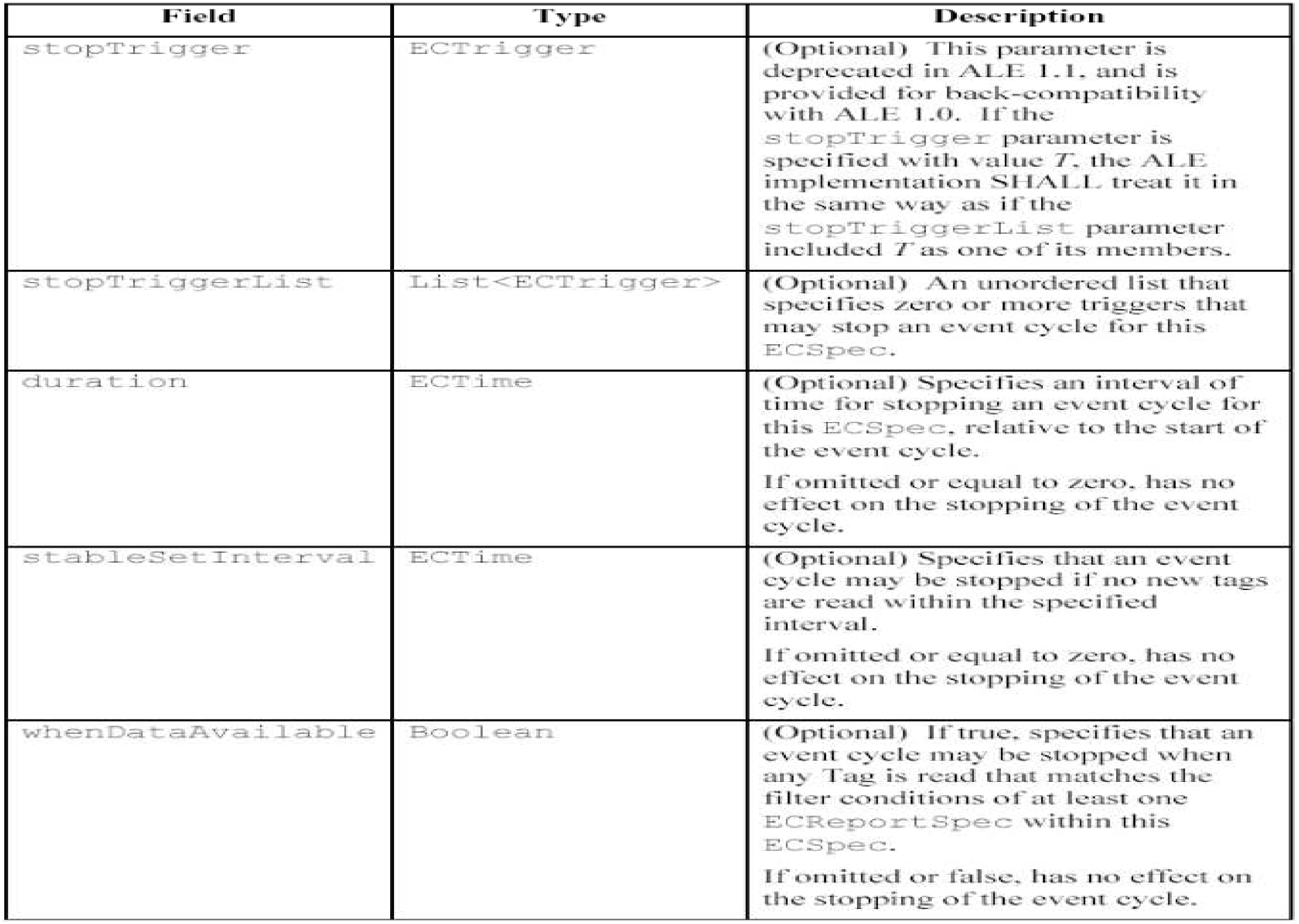
# ECBoundarySpec (2)



| Field | Type | Description |
|---|---|---|
| startTrigger | ECTrigger | (Optional) This parameter is deprecated in ALE 1.1, and is provided for back-compatibility with ALE 1.0. If the startTrigger parameter is specified with value *T*, the ALE implementation SHALL treat it in the same way as if the startTriggerList parameter included *T* as one of its members. |
| startTriggerList | List<ECTrigger> | (Optional) An unordered list that specifies zero or more triggers that may start a new event cycle for this ECSpec. |
| repeatPeriod | ECTime | (Optional) Specifies an interval of time for starting a new event cycle for this ECSpec, relative to the start of the previous event cycle. |



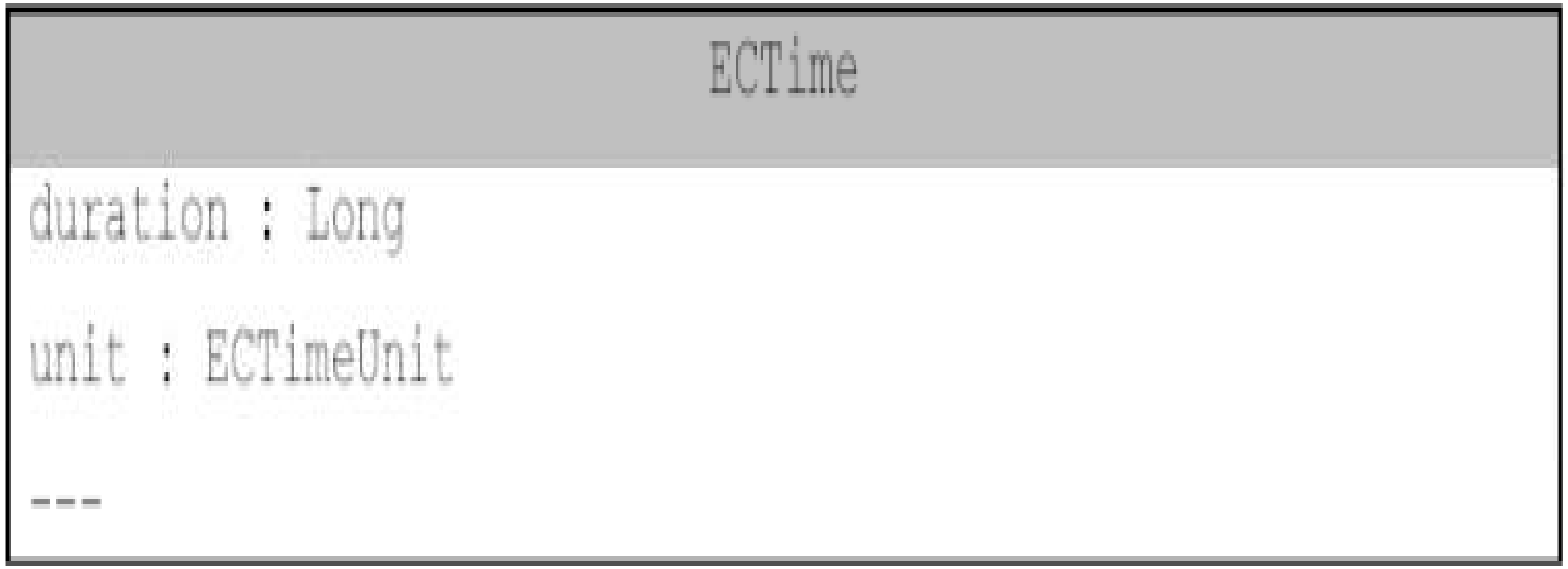
# ECBoundarySpec (3)



| Field | Type | Description |
|---|---|---|
| stopTrigger | ECTrigger | (Optional) This parameter is deprecated in ALE 1.1, and is provided for back-compatibility with ALE 1.0. If the stopTrigger parameter is specified with value *T*, the ALE implementation SHALL treat it in the same way as if the stopTriggerList parameter included *T* as one of its members. |
| stopTriggerList | List<ECTrigger> | (Optional) An unordered list that specifies zero or more triggers that may stop an event cycle for this ECSpec. |
| duration | ECTime | (Optional) Specifies an interval of time for stopping an event cycle for this ECSpec, relative to the start of the event cycle. If omitted or equal to zero, has no effect on the stopping of the event cycle. |
| stableSetInterval | ECTime | (Optional) Specifies that an event cycle may be stopped if no new tags are read within the specified interval. If omitted or equal to zero, has no effect on the stopping of the event cycle. |
| whenDataAvailable | Boolean | (Optional) If true, specifies that an event cycle may be stopped when any Tag is read that matches the filter conditions of at least one ECReportSpec within this ECSpec. If omitted or false, has no effect on the stopping of the event cycle. |

Table 32. ECBoundarySpec Fields

## ECTime (1)

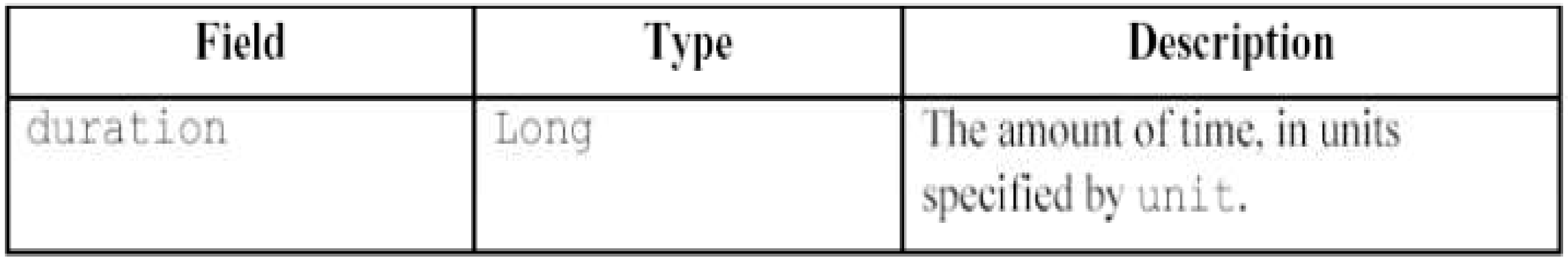

ECTime

duration : Long

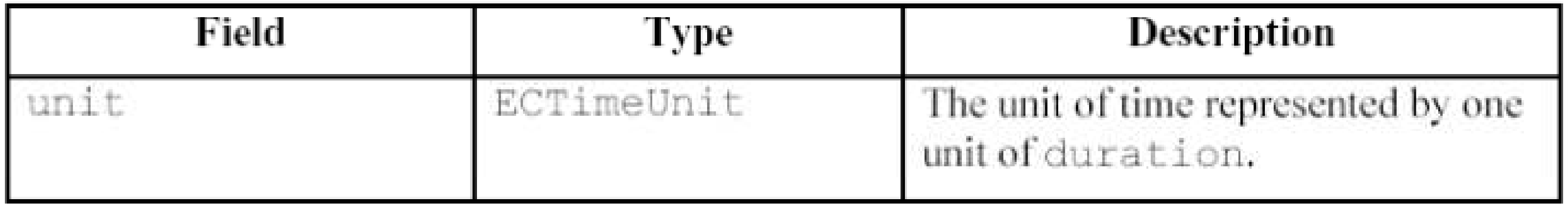
unit : ECTimeUnit

---

## ECTime (2)



| Field | Type | Description |
|---|---|---|
| `duration` | `Long` | The amount of time, in units specified by `unit`. |



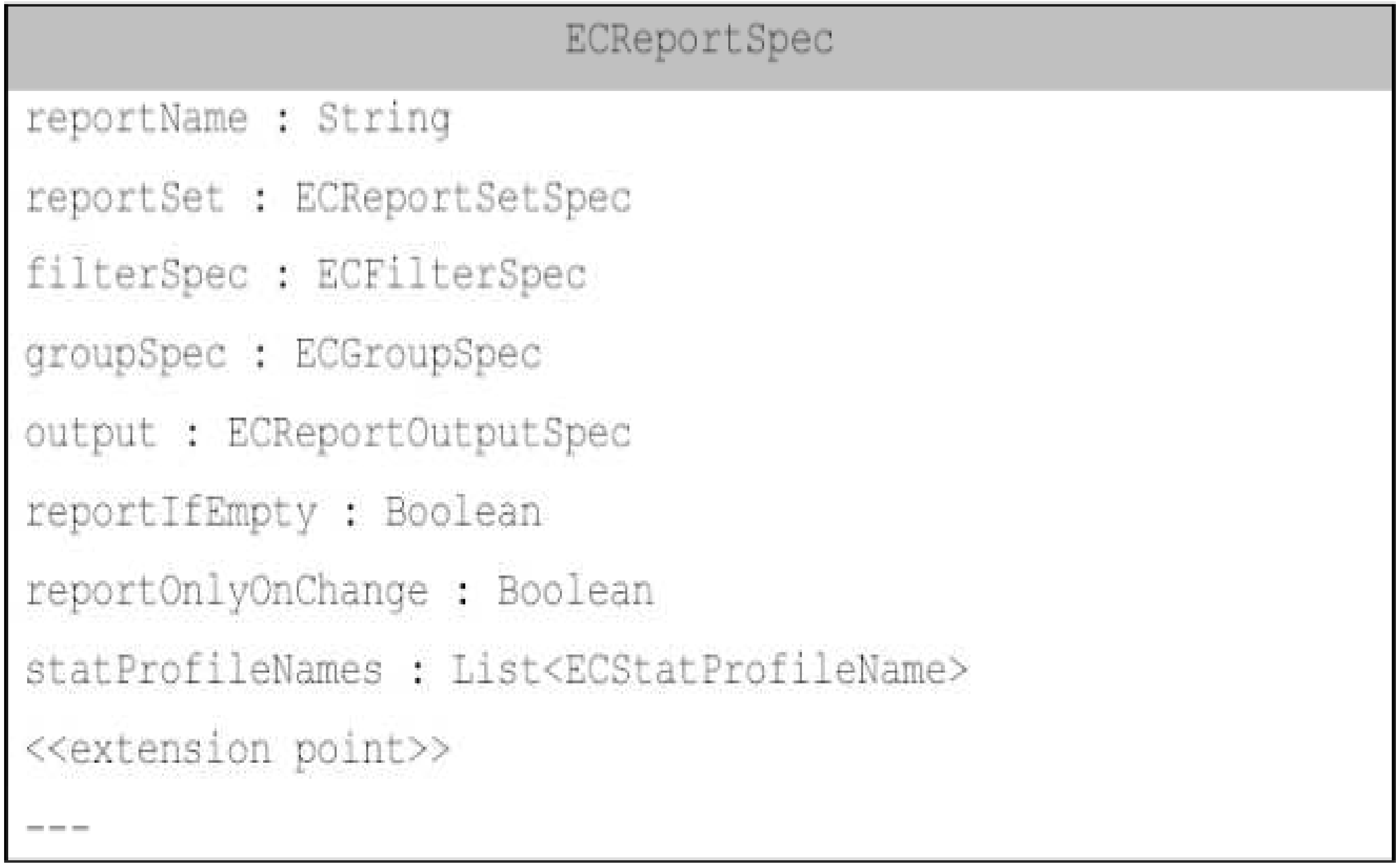
| Field | Type | Description |
|---|---|---|
| `unit` | `ECTimeUnit` | The unit of time represented by one unit of `duration`. |

Table 33. `ECTime` Fields

## ECTrigger

- ECTrigger denotes a URI that is used to specify a start or stop trigger for an event cycle or command cycle
- URIs that begin with the string urn:epcglobal
- They are reserved for standardized trigger URIs whose meaning is governed by this or other EPCglobal specifications

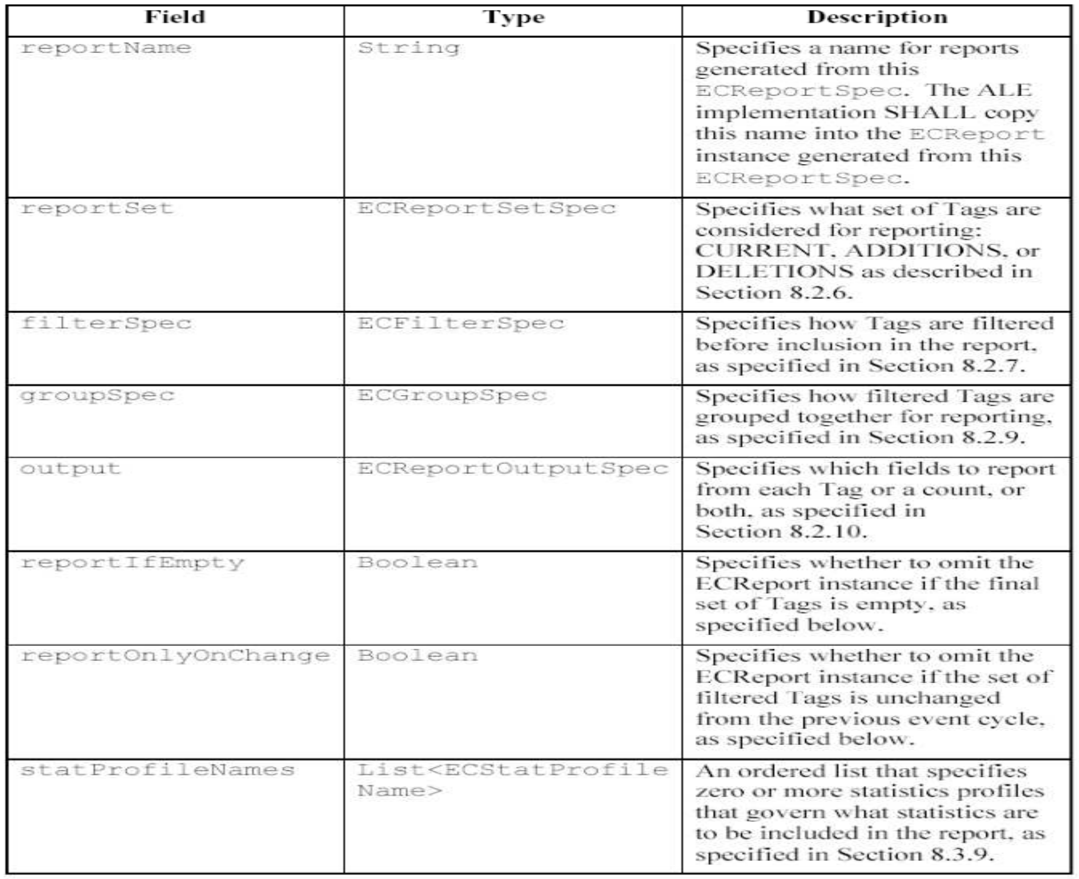
# ECReportSpec (1)



```
ECReportSpec

reportName : String
reportSet : ECReportSetSpec
filterSpec : ECFilterSpec
groupSpec : ECGroupSpec
output : ECReportOutputSpec
reportIfEmpty : Boolean
reportOnlyOnChange : Boolean
statProfileNames : List<ECStatProfileName>
<<extension point>>
---
```

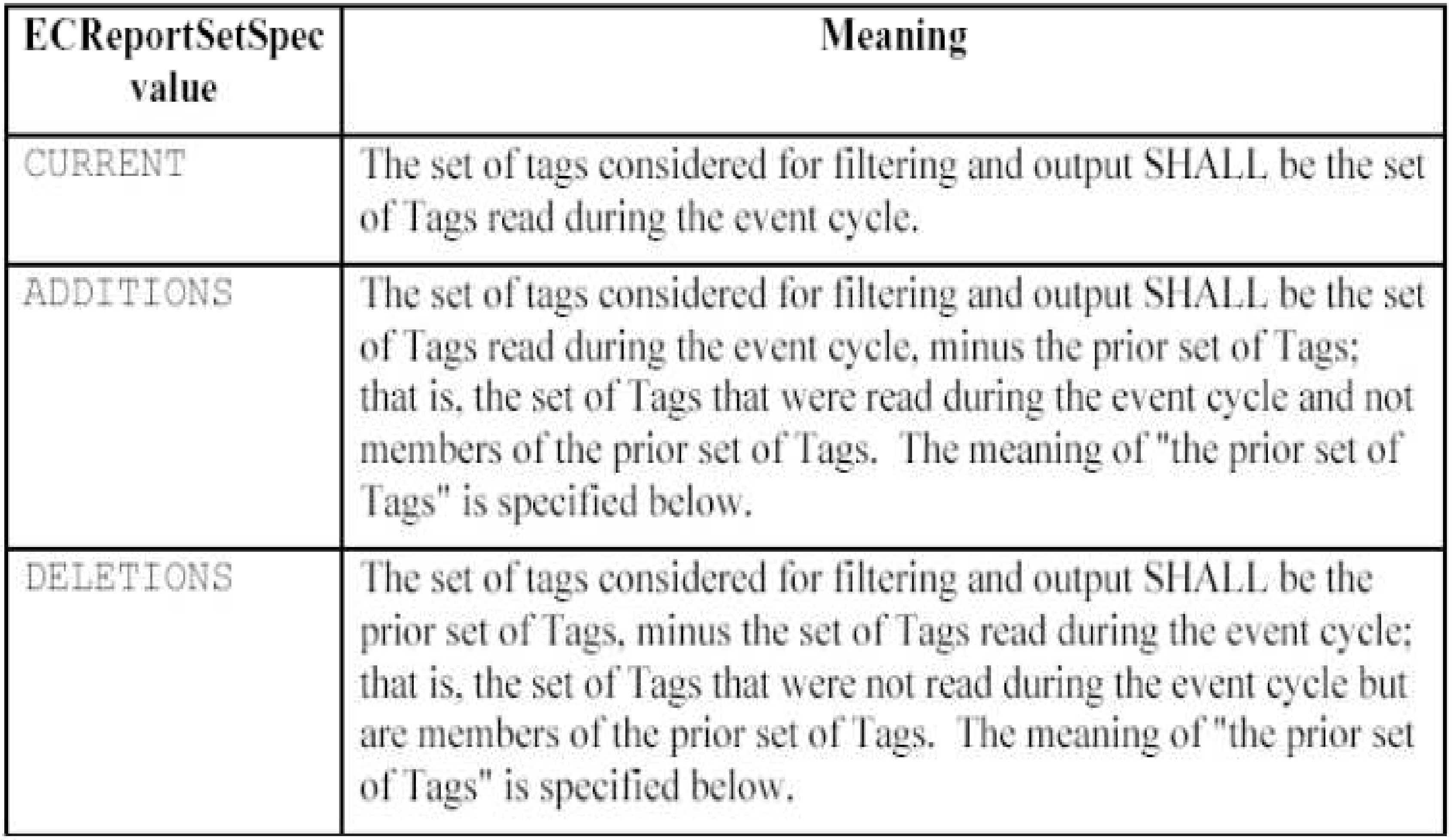
# ECReportSpec (2)



| Field | Type | Description |
|---|---|---|
| reportName | String | Specifies a name for reports generated from this ECReportSpec. The ALE implementation SHALL copy this name into the ECReport instance generated from this ECReportSpec. |
| reportSet | ECReportSetSpec | Specifies what set of Tags are considered for reporting: CURRENT, ADDITIONS, or DELETIONS as described in Section 8.2.6. |
| filterSpec | ECFilterSpec | Specifies how Tags are filtered before inclusion in the report, as specified in Section 8.2.7. |
| groupSpec | ECGroupSpec | Specifies how filtered Tags are grouped together for reporting, as specified in Section 8.2.9. |
| output | ECReportOutputSpec | Specifies which fields to report from each Tag or a count, or both, as specified in Section 8.2.10. |
| reportIfEmpty | Boolean | Specifies whether to omit the ECReport instance if the final set of Tags is empty, as specified below. |
| reportOnlyOnChange | Boolean | Specifies whether to omit the ECReport instance if the set of filtered Tags is unchanged from the previous event cycle, as specified below. |
| statProfileNames | List<ECStatProfileName> | An ordered list that specifies zero or more statistics profiles that govern what statistics are to be included in the report, as specified in Section 8.3.9. |

Table 36. ECReportSpec Fields

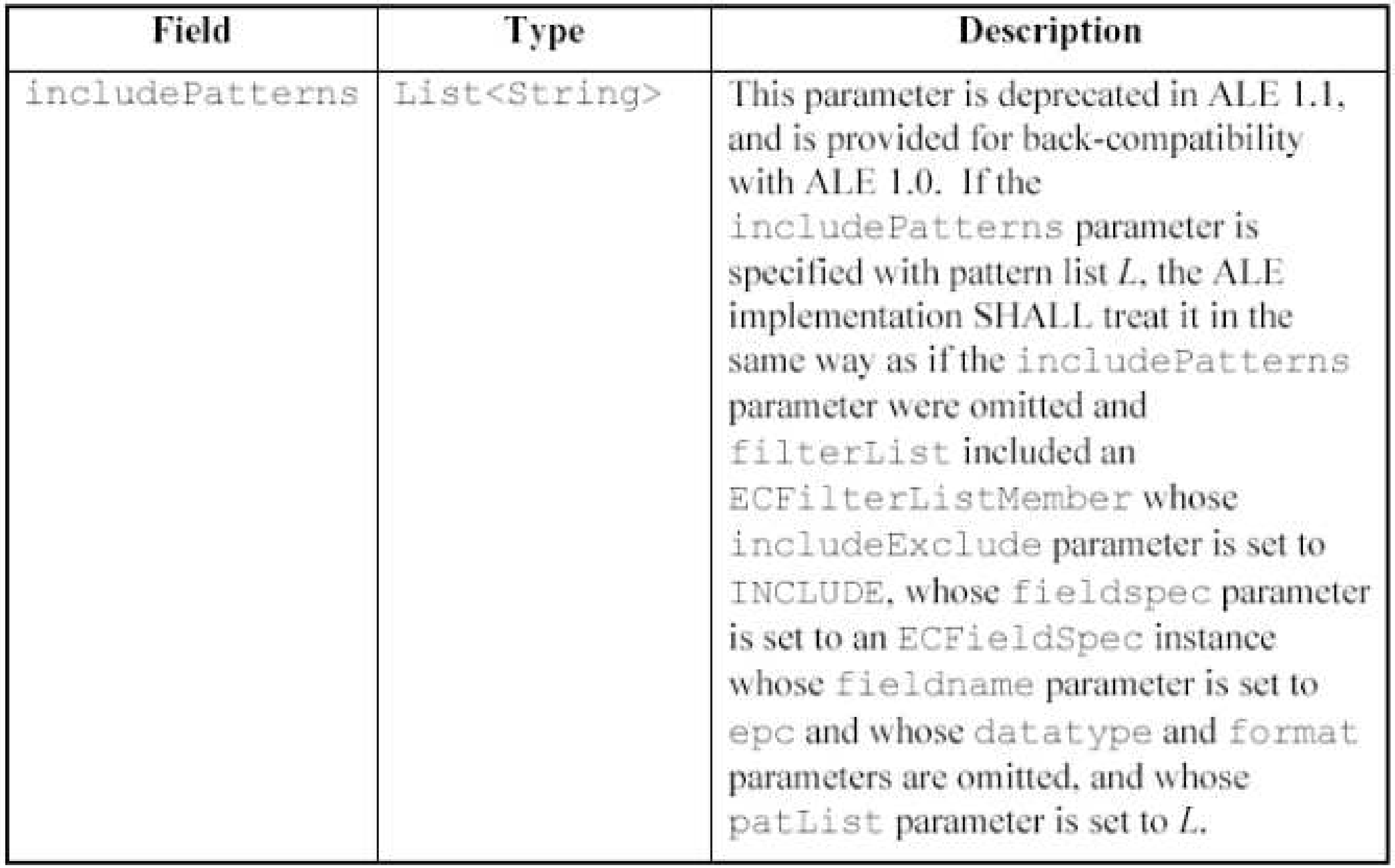
## ECReportSetSpec (3)



| ECReportSetSpec value | Meaning |
|---|---|
| CURRENT | The set of tags considered for filtering and output SHALL be the set of Tags read during the event cycle. |
| ADDITIONS | The set of tags considered for filtering and output SHALL be the set of Tags read during the event cycle, minus the prior set of Tags; that is, the set of Tags that were read during the event cycle and not members of the prior set of Tags. The meaning of "the prior set of Tags" is specified below. |
| DELETIONS | The set of tags considered for filtering and output SHALL be the prior set of Tags, minus the set of Tags read during the event cycle; that is, the set of Tags that were not read during the event cycle but are members of the prior set of Tags. The meaning of "the prior set of Tags" is specified below. |

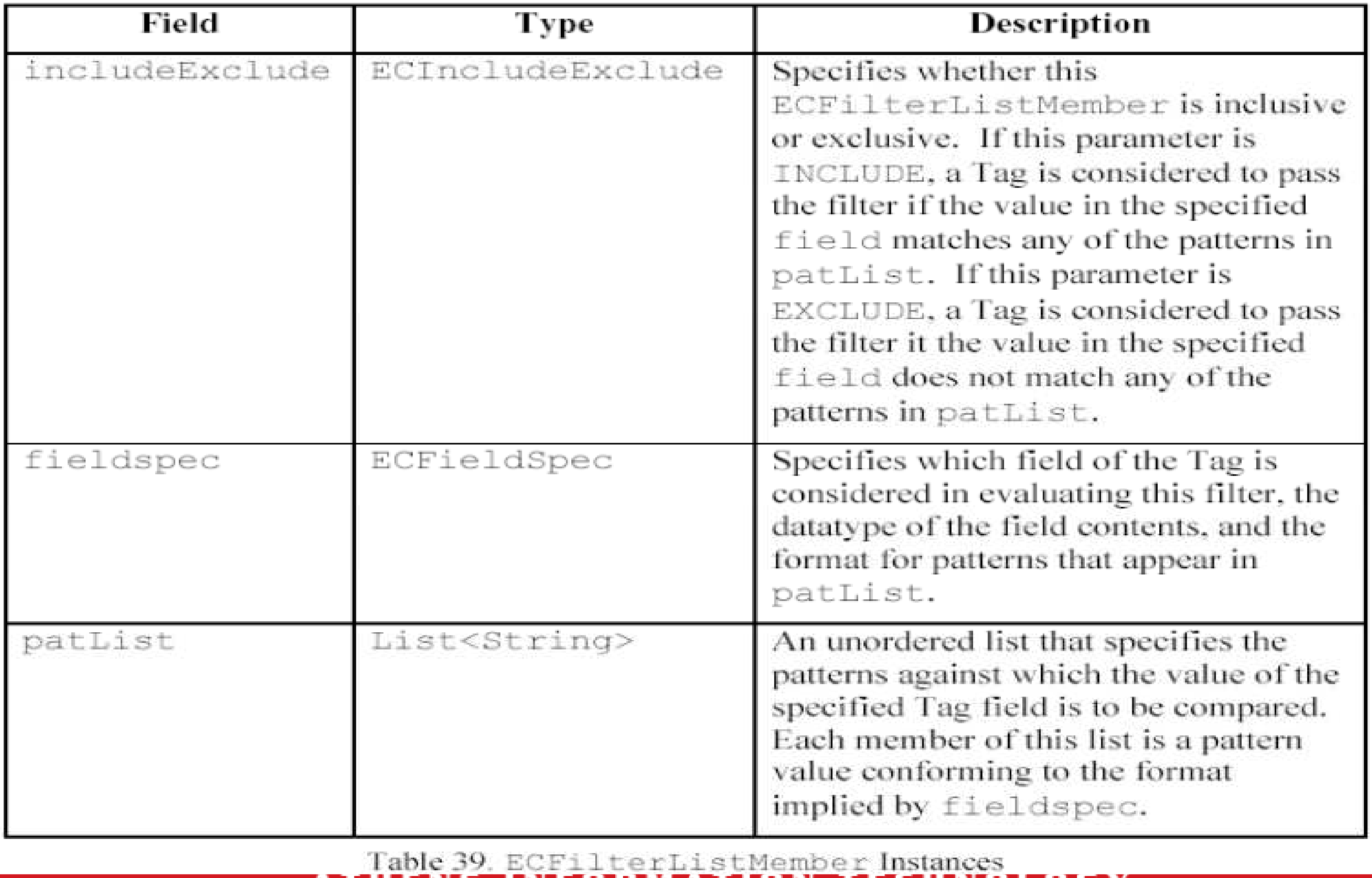
Table 37. ECReportSetSpec Values

# ECFilterSpec



| Field | Type | Description |
|---|---|---|
| `includePatterns` | `List<String>` | This parameter is deprecated in ALE 1.1, and is provided for back-compatibility with ALE 1.0. If the `includePatterns` parameter is specified with pattern list *L*, the ALE implementation SHALL treat it in the same way as if the `includePatterns` parameter were omitted and `filterList` included an `ECFilterListMember` whose `includeExclude` parameter is set to `INCLUDE`, whose `fieldspec` parameter is set to an `ECFieldSpec` instance whose `fieldname` parameter is set to `epc` and whose `datatype` and `format` parameters are omitted, and whose `patList` parameter is set to *L*. |

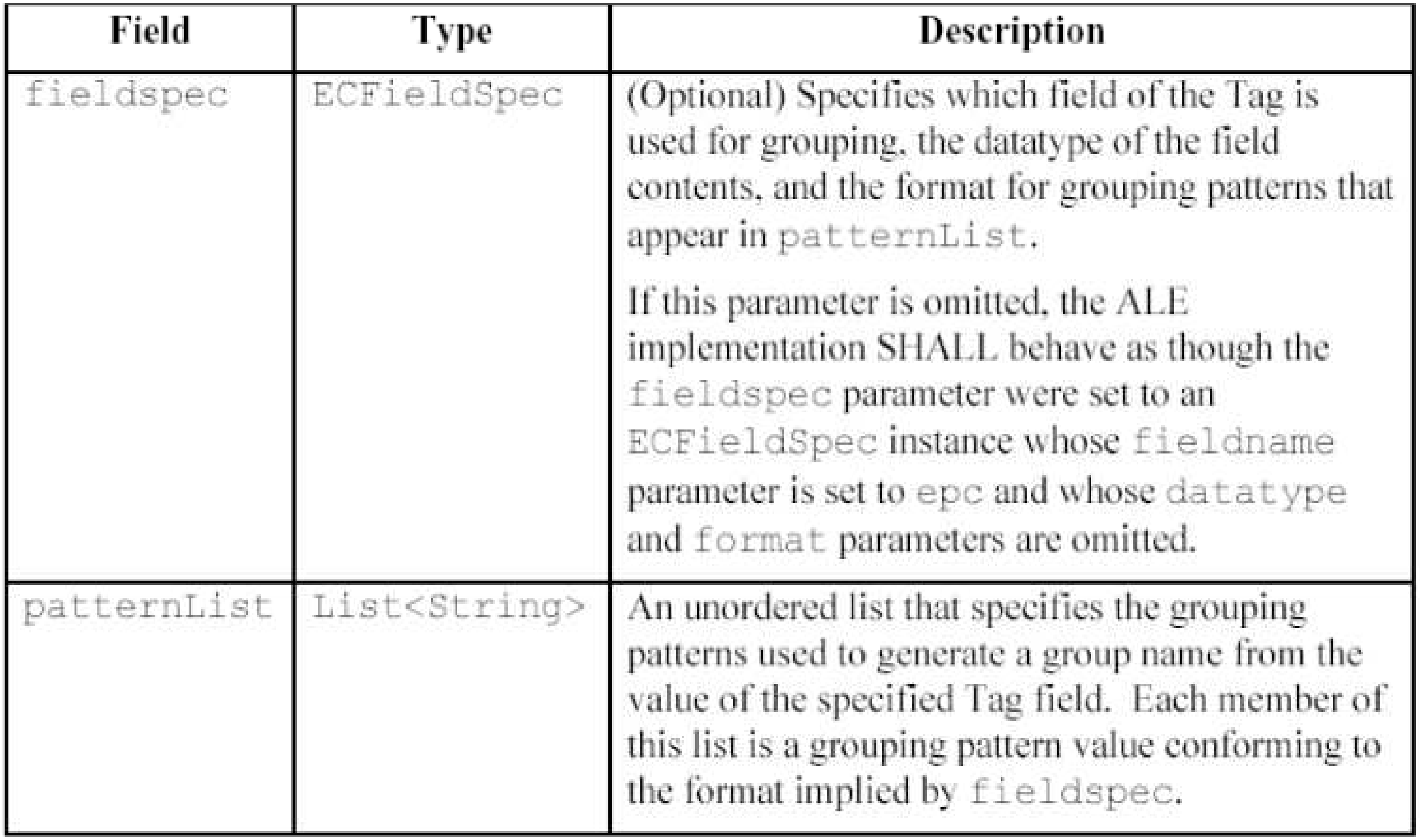
# ECFilterListMember



| Field | Type | Description |
|---|---|---|
| includeExclude | ECIncludeExclude | Specifies whether this ECFilterListMember is inclusive or exclusive. If this parameter is INCLUDE, a Tag is considered to pass the filter if the value in the specified field matches any of the patterns in patList. If this parameter is EXCLUDE, a Tag is considered to pass the filter it the value in the specified field does not match any of the patterns in patList. |
| fieldspec | ECFieldSpec | Specifies which field of the Tag is considered in evaluating this filter, the datatype of the field contents, and the format for patterns that appear in patList. |
| patList | List<String> | An unordered list that specifies the patterns against which the value of the specified Tag field is to be compared. Each member of this list is a pattern value conforming to the format implied by fieldspec. |

Table 39. ECFilterListMember Instances

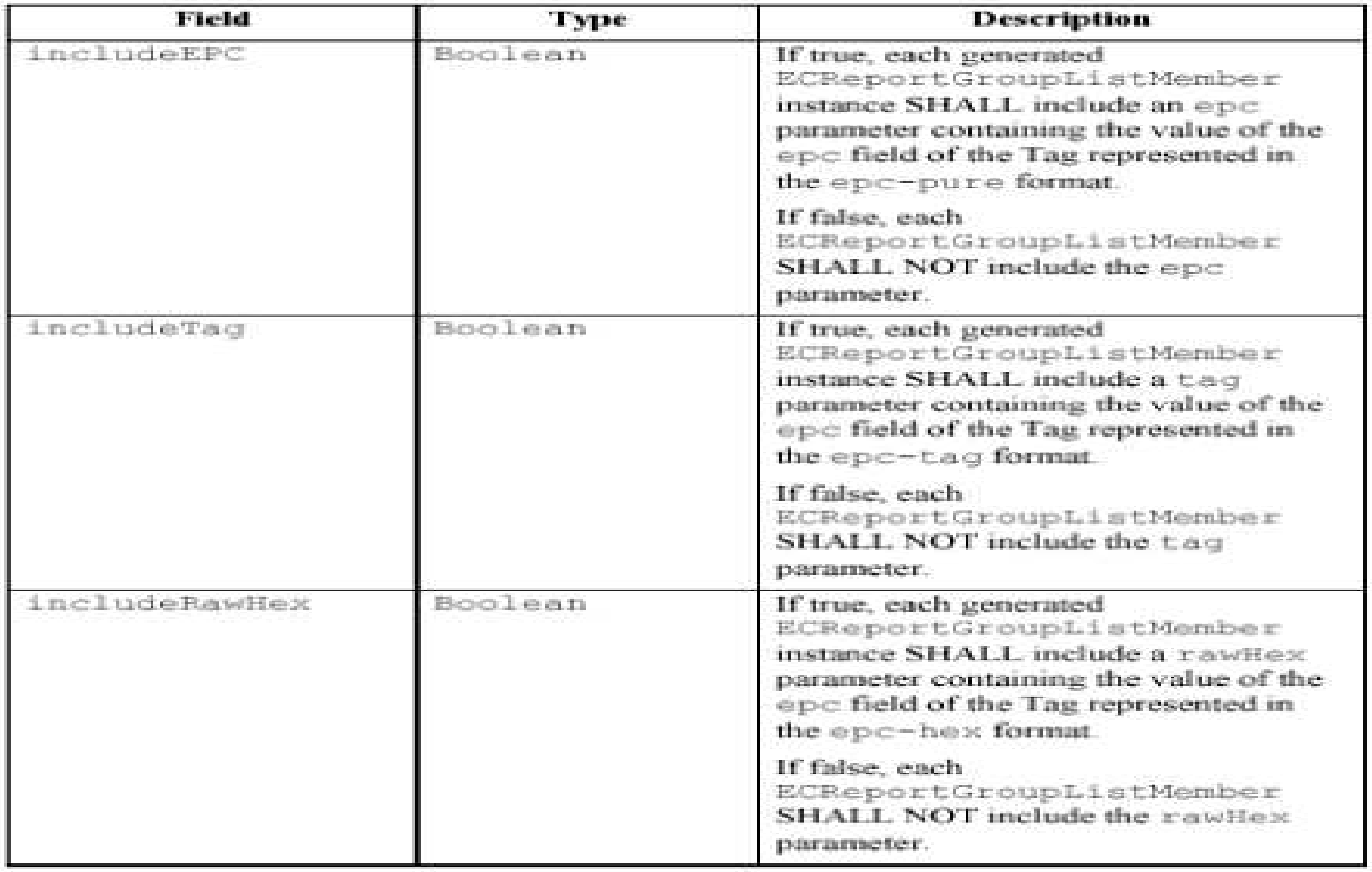
## ECGroupSpec



| Field | Type | Description |
|---|---|---|
| fieldspec | ECFieldSpec | (Optional) Specifies which field of the Tag is used for grouping, the datatype of the field contents, and the format for grouping patterns that appear in patternList.<br><br>If this parameter is omitted, the ALE implementation SHALL behave as though the fieldspec parameter were set to an ECFieldSpec instance whose fieldname parameter is set to epc and whose datatype and format parameters are omitted. |
| patternList | List<String> | An unordered list that specifies the grouping patterns used to generate a group name from the value of the specified Tag field. Each member of this list is a grouping pattern value conforming to the format implied by fieldspec. |

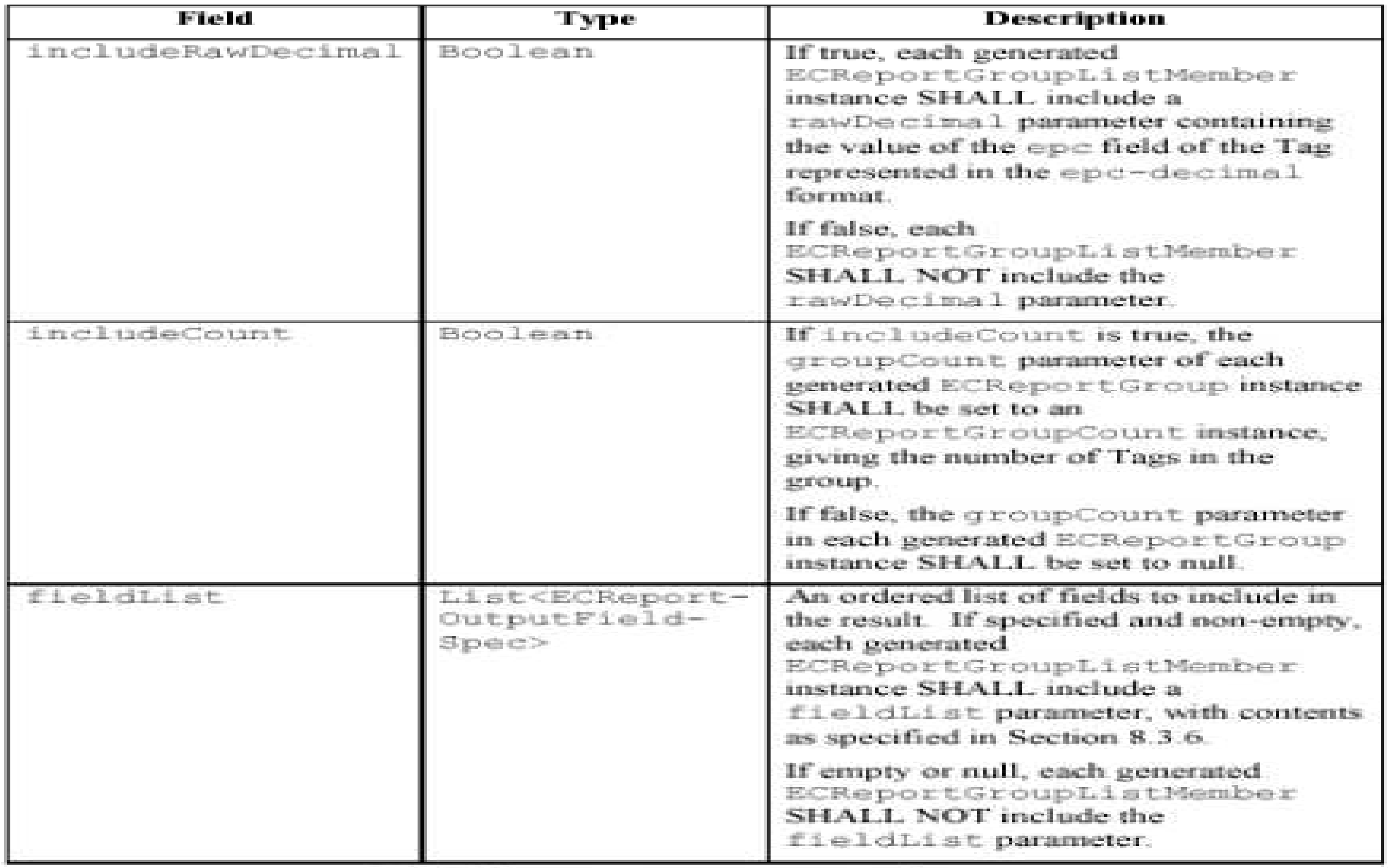
Table 40. ECGroupSpec Fields

# ECReportOutputSpec (1)



| Field | Type | Description |
|---|---|---|
| includeEPC | Boolean | If true, each generated ECReportGroupListMember instance SHALL include an epc parameter containing the value of the epc field of the Tag represented in the epc-pure format.<br>If false, each ECReportGroupListMember SHALL NOT include the epc parameter. |
| includeTag | Boolean | If true, each generated ECReportGroupListMember instance SHALL include a tag parameter containing the value of the epc field of the Tag represented in the epc-tag format.<br>If false, each ECReportGroupListMember SHALL NOT include the tag parameter. |
| includeRawHex | Boolean | If true, each generated ECReportGroupListMember instance SHALL include a rawHex parameter containing the value of the epc field of the Tag represented in the epc-hex format.<br>If false, each ECReportGroupListMember SHALL NOT include the rawHex parameter. |

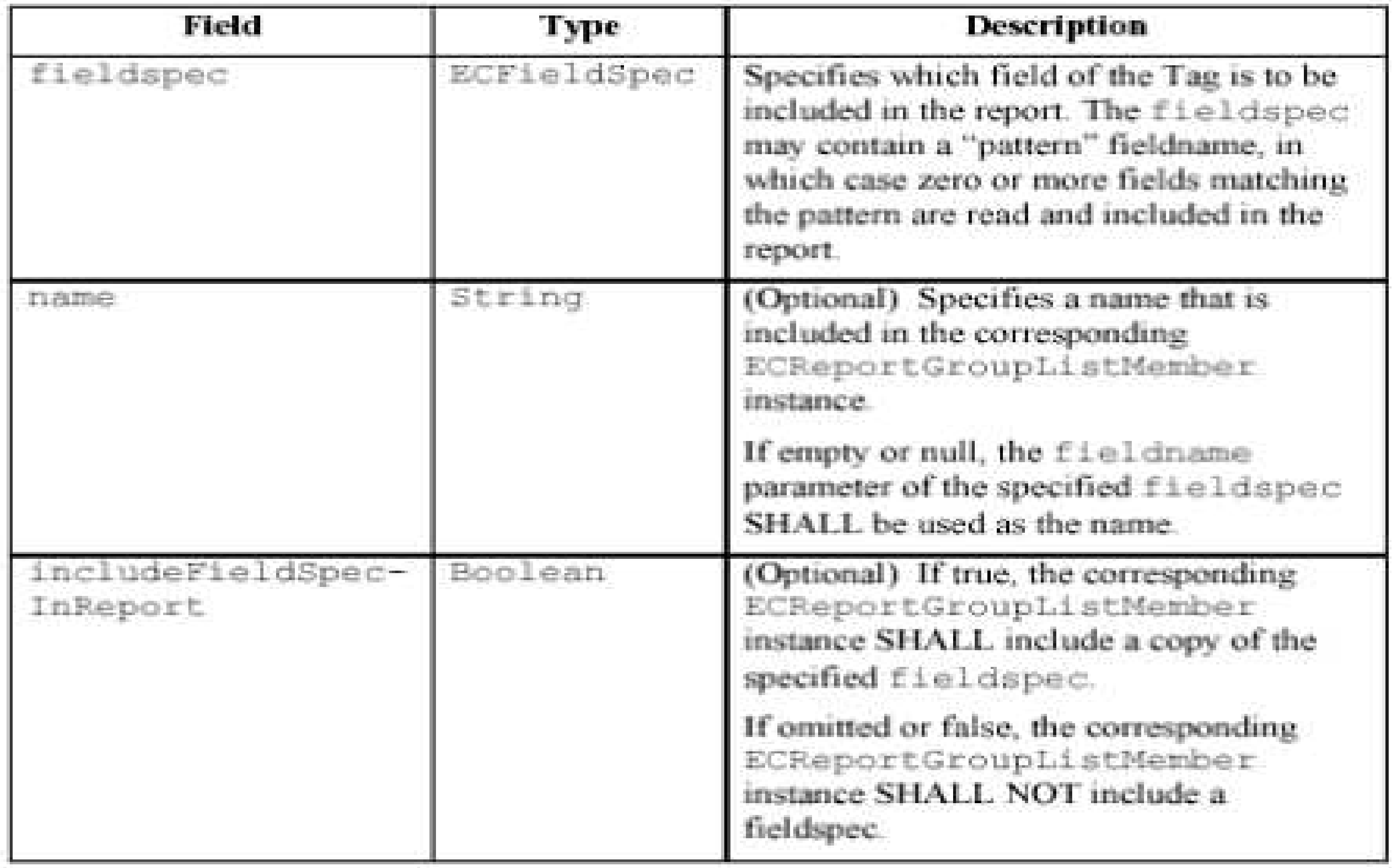
# ECReportOutputSpec (2)



| Field | Type | Description |
|---|---|---|
| includeRawDecimal | Boolean | If true, each generated ECReportGroupListMember instance SHALL include a rawDecimal parameter containing the value of the epc field of the Tag represented in the epc-decimal format.<br><br>If false, each ECReportGroupListMember SHALL NOT include the rawDecimal parameter. |
| includeCount | Boolean | If includeCount is true, the groupCount parameter of each generated ECReportGroup instance SHALL be set to an ECReportGroupCount instance, giving the number of Tags in the group.<br><br>If false, the groupCount parameter in each generated ECReportGroup instance SHALL be set to null. |
| fieldList | List<ECReport-OutputField-Spec> | An ordered list of fields to include in the result. If specified and non-empty, each generated ECReportGroupListMember instance SHALL include a fieldList parameter, with contents as specified in Section 8.3.6.<br><br>If empty or null, each generated ECReportGroupListMember SHALL NOT include the fieldList parameter. |

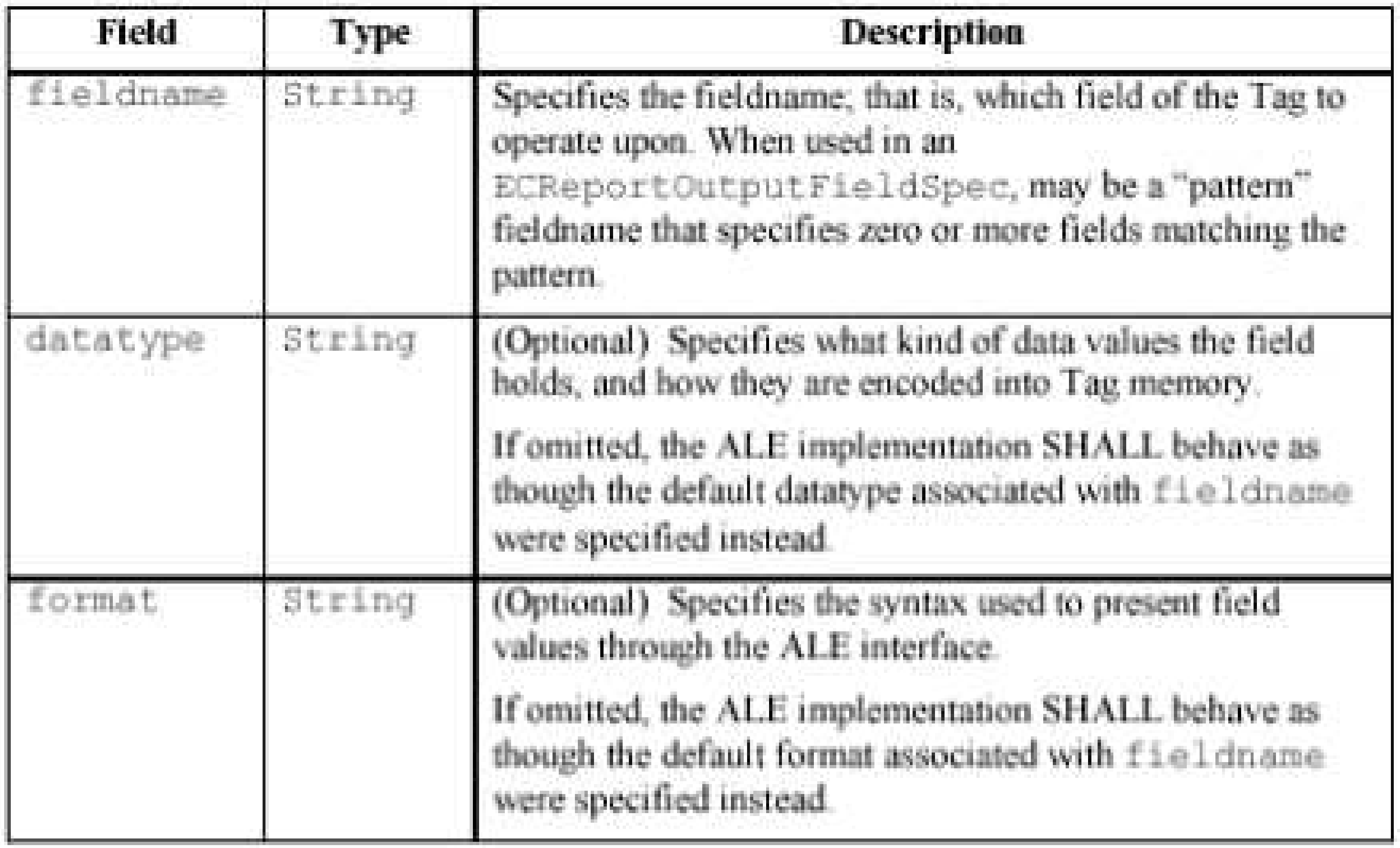
Table 41. ECReportOutputSpec Instance

# ECReportOutputFieldSpec



| Field | Type | Description |
|---|---|---|
| `fieldspec` | `ECFieldSpec` | Specifies which field of the Tag is to be included in the report. The `fieldspec` may contain a "pattern" fieldname, in which case zero or more fields matching the pattern are read and included in the report. |
| `name` | `String` | (Optional) Specifies a name that is included in the corresponding `ECReportGroupListMember` instance.<br><br>If empty or null, the `fieldname` parameter of the specified `fieldspec` SHALL be used as the name. |
| `includeFieldSpec-InReport` | `Boolean` | (Optional) If true, the corresponding `ECReportGroupListMember` instance SHALL include a copy of the specified `fieldspec`.<br><br>If omitted or false, the corresponding `ECReportGroupListMember` instance SHALL NOT include a fieldspec. |

# ECFieldSpec

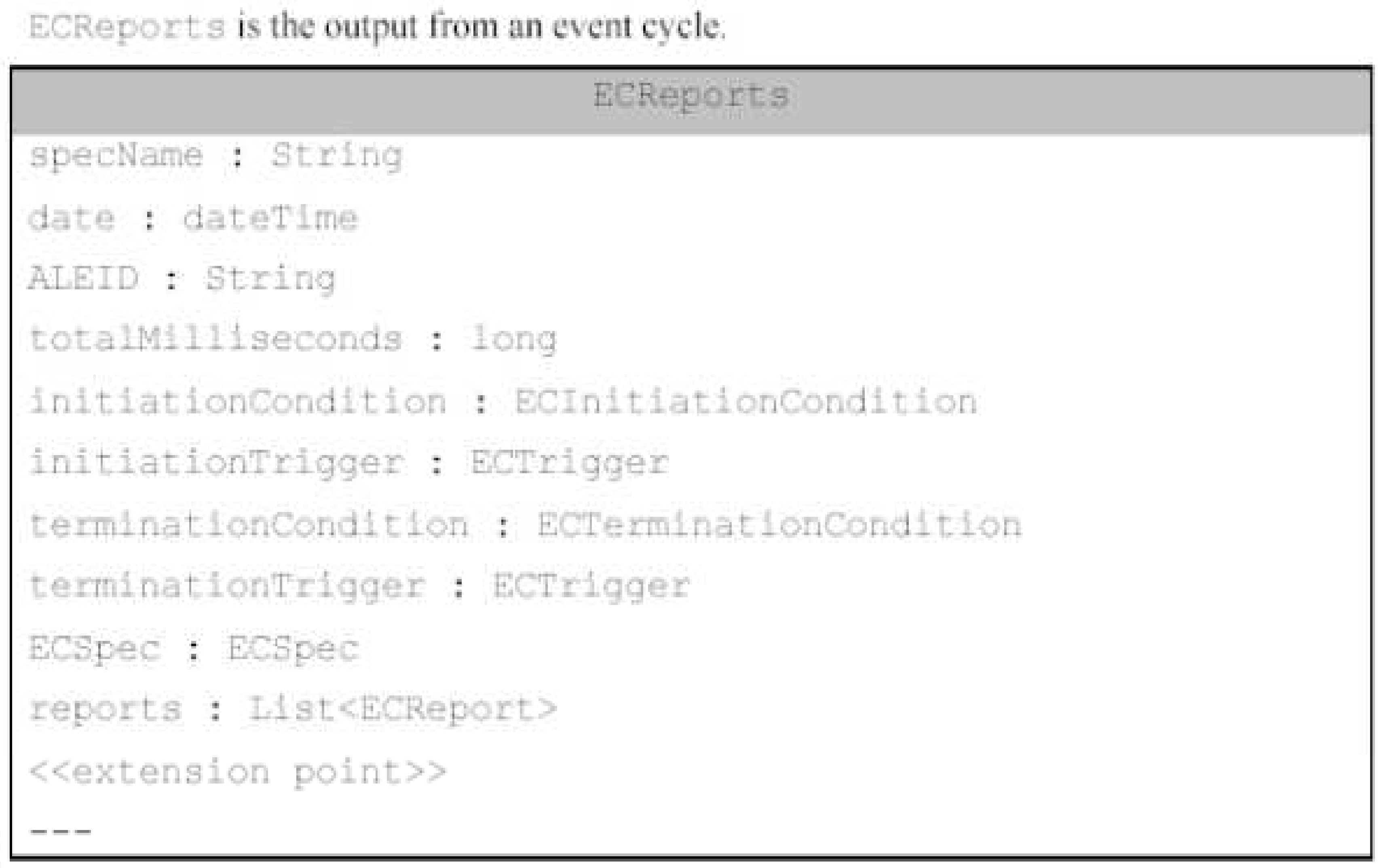


| Field | Type | Description |
|---|---|---|
| `fieldname` | `String` | Specifies the fieldname; that is, which field of the Tag to operate upon. When used in an `ECReportOutputFieldSpec`, may be a "pattern" fieldname that specifies zero or more fields matching the pattern. |
| `datatype` | `String` | (Optional) Specifies what kind of data values the field holds, and how they are encoded into Tag memory.<br><br>If omitted, the ALE implementation SHALL behave as though the default datatype associated with `fieldname` were specified instead. |
| `format` | `String` | (Optional) Specifies the syntax used to present field values through the ALE interface.<br><br>If omitted, the ALE implementation SHALL behave as though the default format associated with `fieldname` were specified instead. |

Table 43. `ECFieldSpec` Fields

200

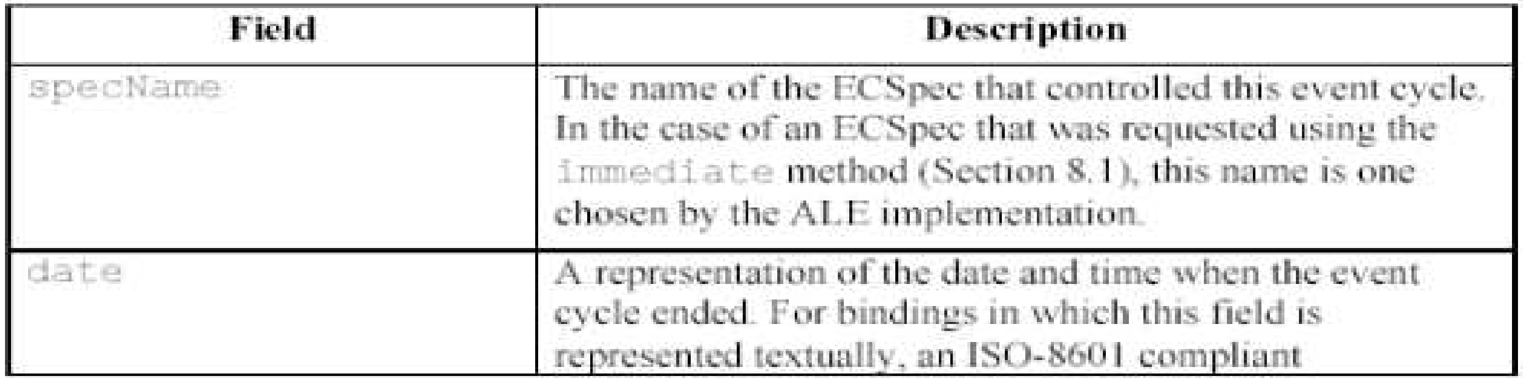
# ECReports (1)



ECReports is the output from an event cycle.

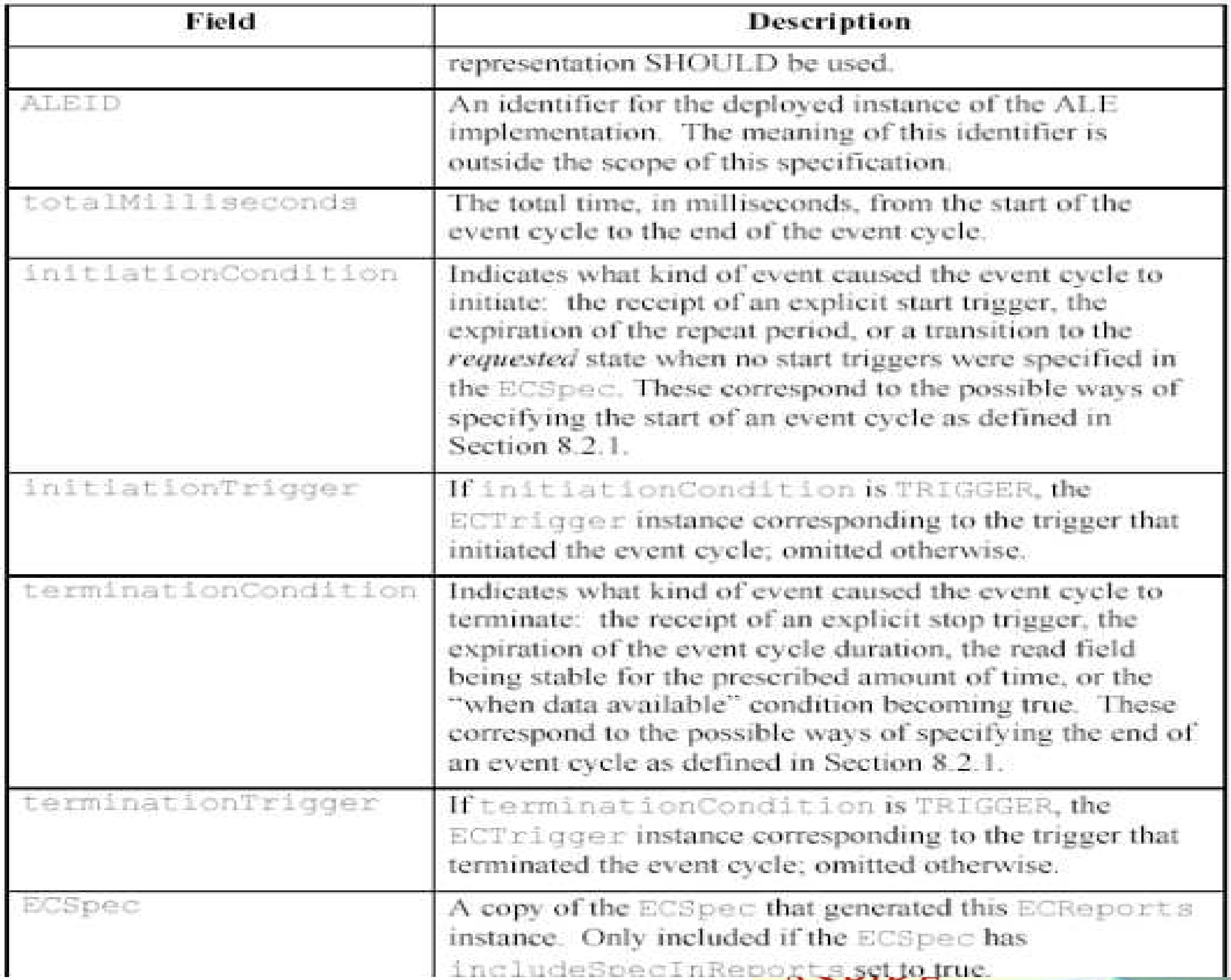
| ECReports |
|---|
| specName : String |
| date : dateTime |
| ALEID : String |
| totalMilliseconds : long |
| initiationCondition : ECInitiationCondition |
| initiationTrigger : ECTrigger |
| terminationCondition : ECTerminationCondition |
| terminationTrigger : ECTrigger |
| ECSpec : ECSpec |
| reports : List<ECReport> |
| <<extension point>> |
| --- |

## ECReports (2)



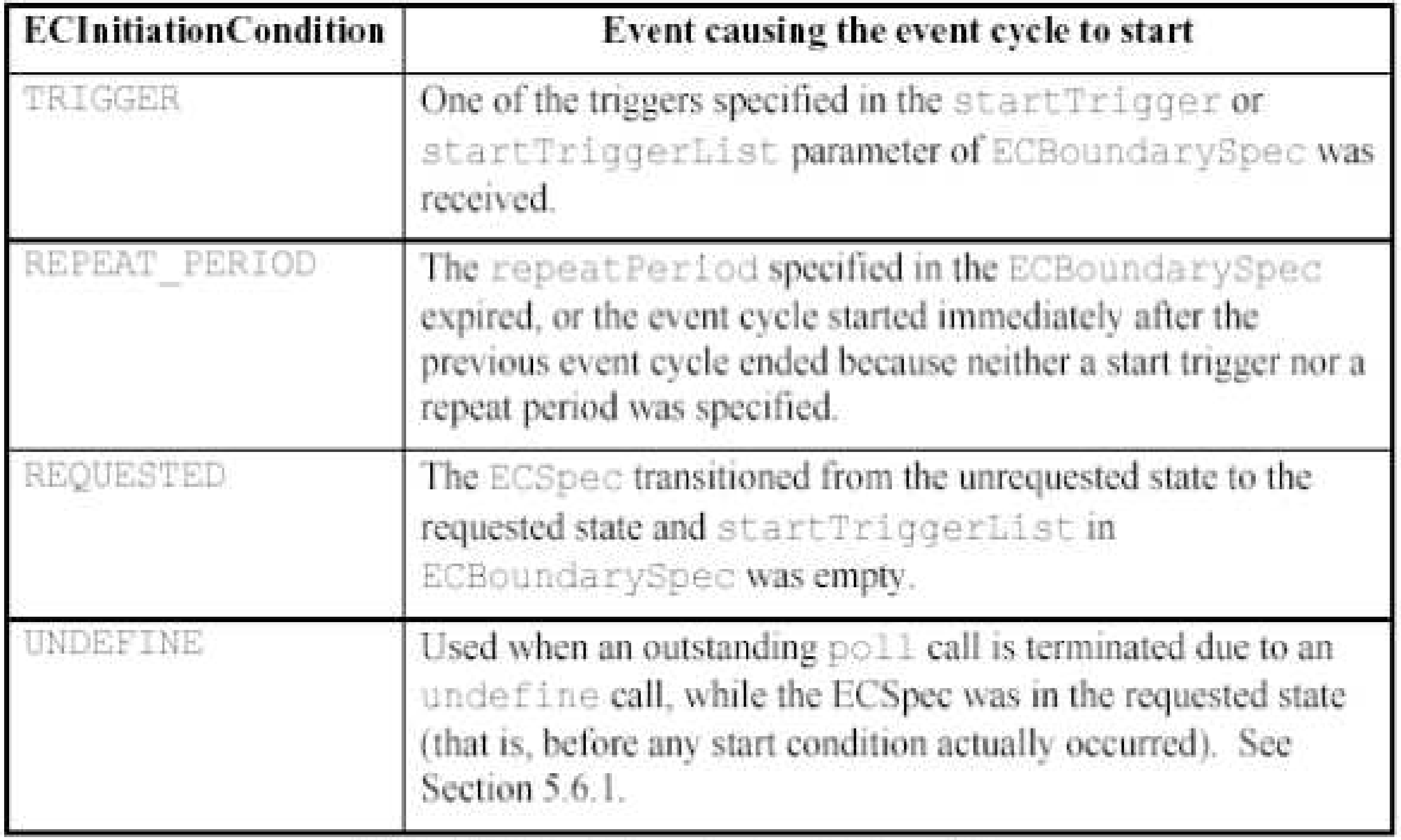
| Field | Description |
|---|---|
| `specName` | The name of the ECSpec that controlled this event cycle. In the case of an ECSpec that was requested using the `immediate` method (Section 8.1), this name is one chosen by the ALE implementation. |
| `date` | A representation of the date and time when the event cycle ended. For bindings in which this field is represented textually, an ISO-8601 compliant |

# ECReports (3)



| Field | Description |
|---|---|
| | representation SHOULD be used. |
| ALEID | An identifier for the deployed instance of the ALE implementation. The meaning of this identifier is outside the scope of this specification. |
| totalMilliseconds | The total time, in milliseconds, from the start of the event cycle to the end of the event cycle. |
| initiationCondition | Indicates what kind of event caused the event cycle to initiate: the receipt of an explicit start trigger, the expiration of the repeat period, or a transition to the *requested* state when no start triggers were specified in the ECSpec. These correspond to the possible ways of specifying the start of an event cycle as defined in Section 8.2.1. |
| initiationTrigger | If initiationCondition is TRIGGER, the ECTrigger instance corresponding to the trigger that initiated the event cycle; omitted otherwise. |
| terminationCondition | Indicates what kind of event caused the event cycle to terminate: the receipt of an explicit stop trigger, the expiration of the event cycle duration, the read field being stable for the prescribed amount of time, or the "when data available" condition becoming true. These correspond to the possible ways of specifying the end of an event cycle as defined in Section 8.2.1. |
| terminationTrigger | If terminationCondition is TRIGGER, the ECTrigger instance corresponding to the trigger that terminated the event cycle; omitted otherwise. |
| ECSpec | A copy of the ECSpec that generated this ECReports instance. Only included if the ECSpec has includeSpecInReports set to true. |

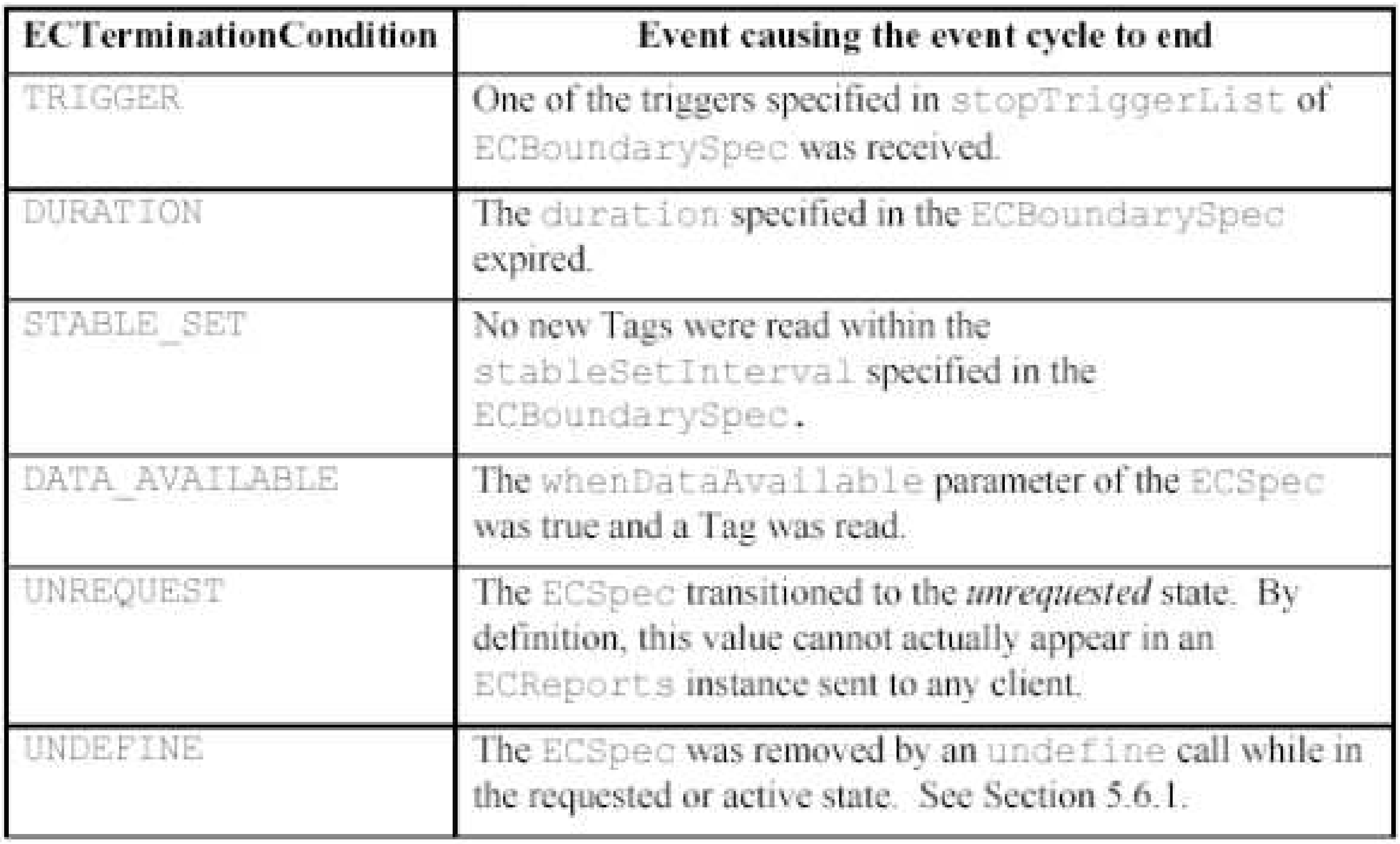
# ECInitiationCondition



| ECInitiationCondition | Event causing the event cycle to start |
|---|---|
| TRIGGER | One of the triggers specified in the startTrigger or startTriggerList parameter of ECBoundarySpec was received. |
| REPEAT_PERIOD | The repeatPeriod specified in the ECBoundarySpec expired, or the event cycle started immediately after the previous event cycle ended because neither a start trigger nor a repeat period was specified. |
| REQUESTED | The ECSpec transitioned from the unrequested state to the requested state and startTriggerList in ECBoundarySpec was empty. |
| UNDEFINE | Used when an outstanding poll call is terminated due to an undefine call, while the ECSpec was in the requested state (that is, before any start condition actually occurred). See Section 5.6.1. |

Table 45. ECInitiationCondition Values

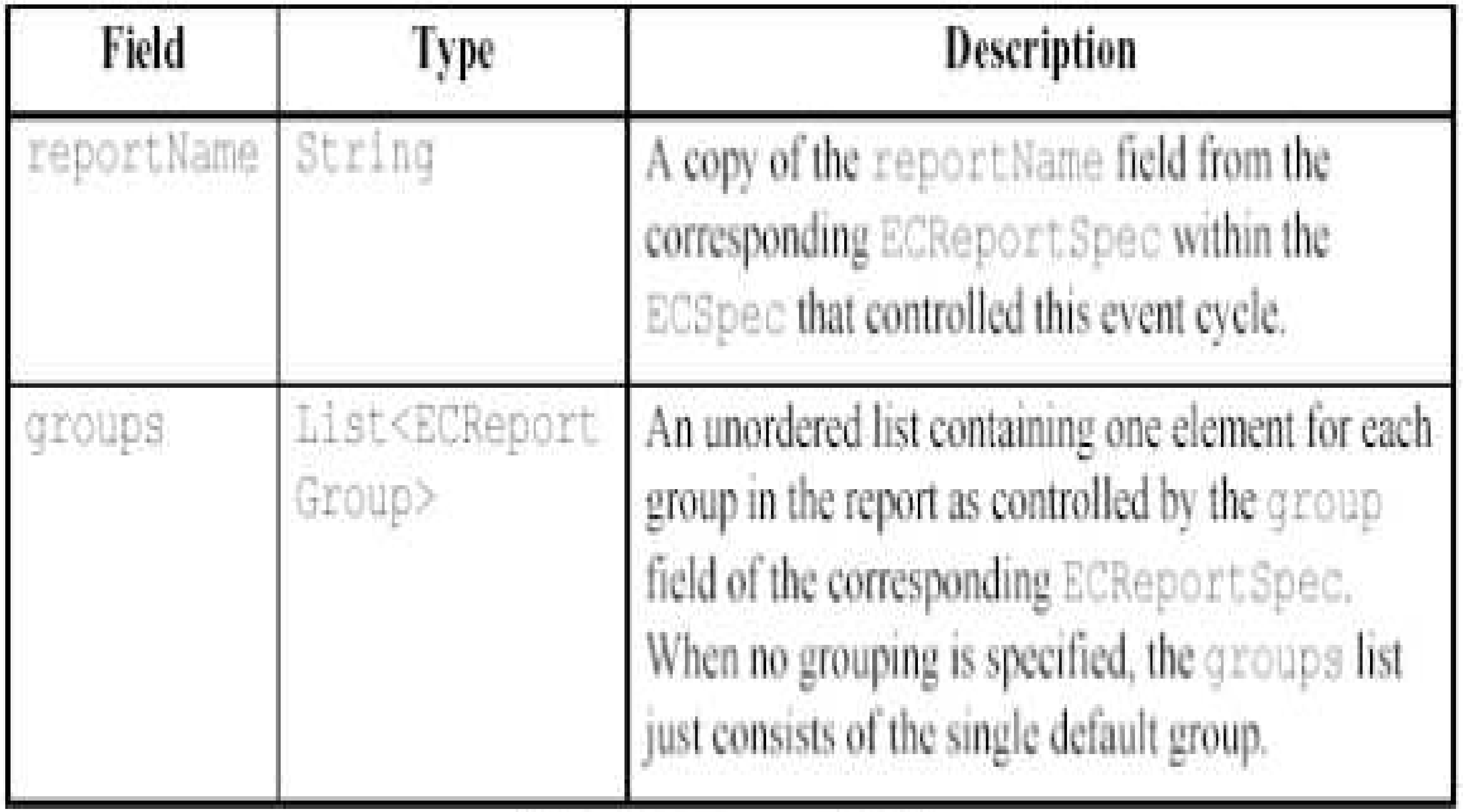
# ECTerminationCondition



| ECTerminationCondition | Event causing the event cycle to end |
|---|---|
| TRIGGER | One of the triggers specified in stopTriggerList of ECBoundarySpec was received. |
| DURATION | The duration specified in the ECBoundarySpec expired. |
| STABLE_SET | No new Tags were read within the stableSetInterval specified in the ECBoundarySpec. |
| DATA_AVAILABLE | The whenDataAvailable parameter of the ECSpec was true and a Tag was read. |
| UNREQUEST | The ECSpec transitioned to the *unrequested* state. By definition, this value cannot actually appear in an ECReports instance sent to any client. |
| UNDEFINE | The ECSpec was removed by an undefine call while in the requested or active state. See Section 5.6.1. |

Table 46. ECTerminationCondition Values

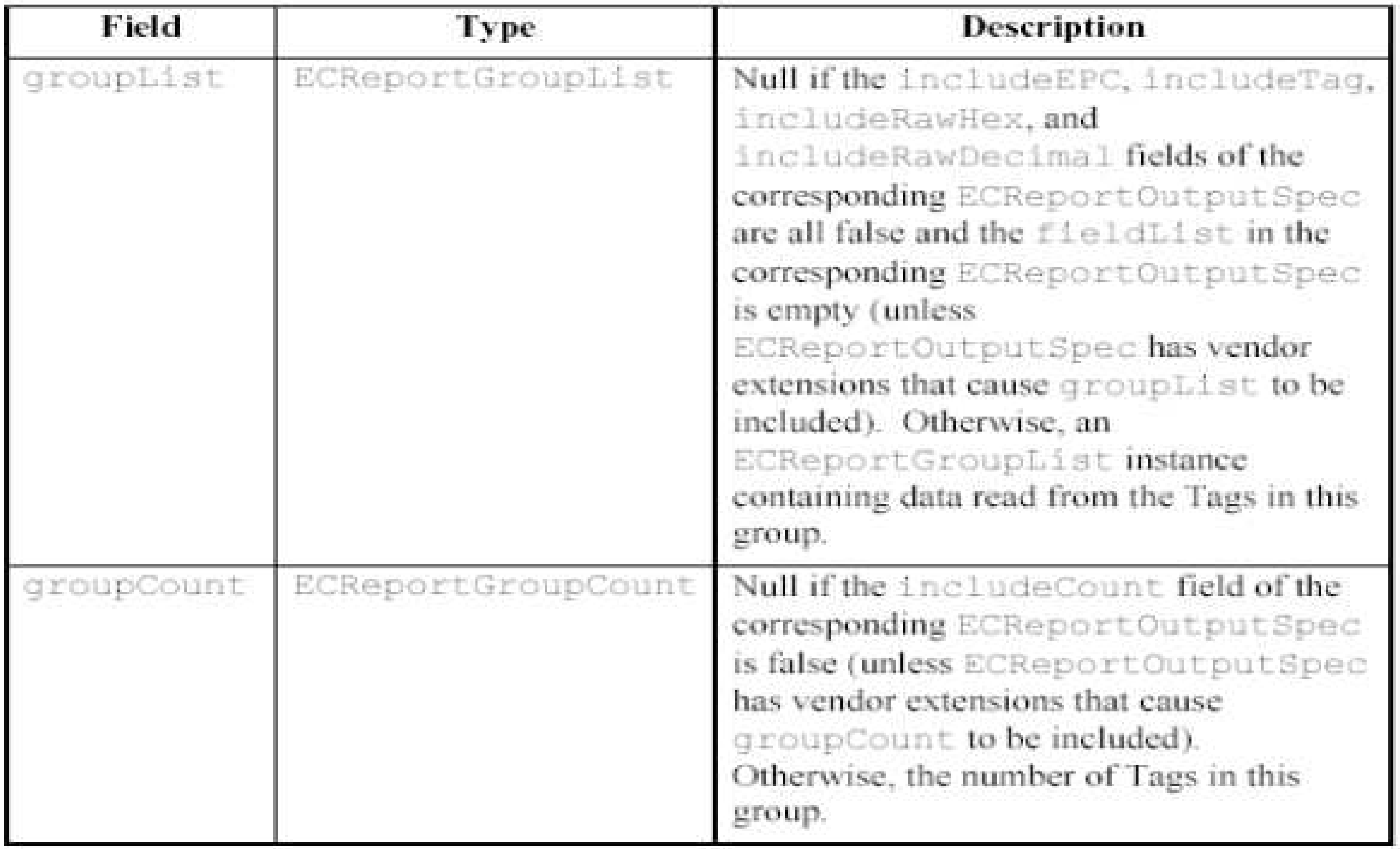
# ECReport



| Field | Type | Description |
|---|---|---|
| reportName | String | A copy of the reportName field from the corresponding ECReportSpec within the ECSpec that controlled this event cycle. |
| groups | List<ECReport Group> | An unordered list containing one element for each group in the report as controlled by the group field of the corresponding ECReportSpec. When no grouping is specified, the groups list just consists of the single default group. |

Table 47. ECReport Fields

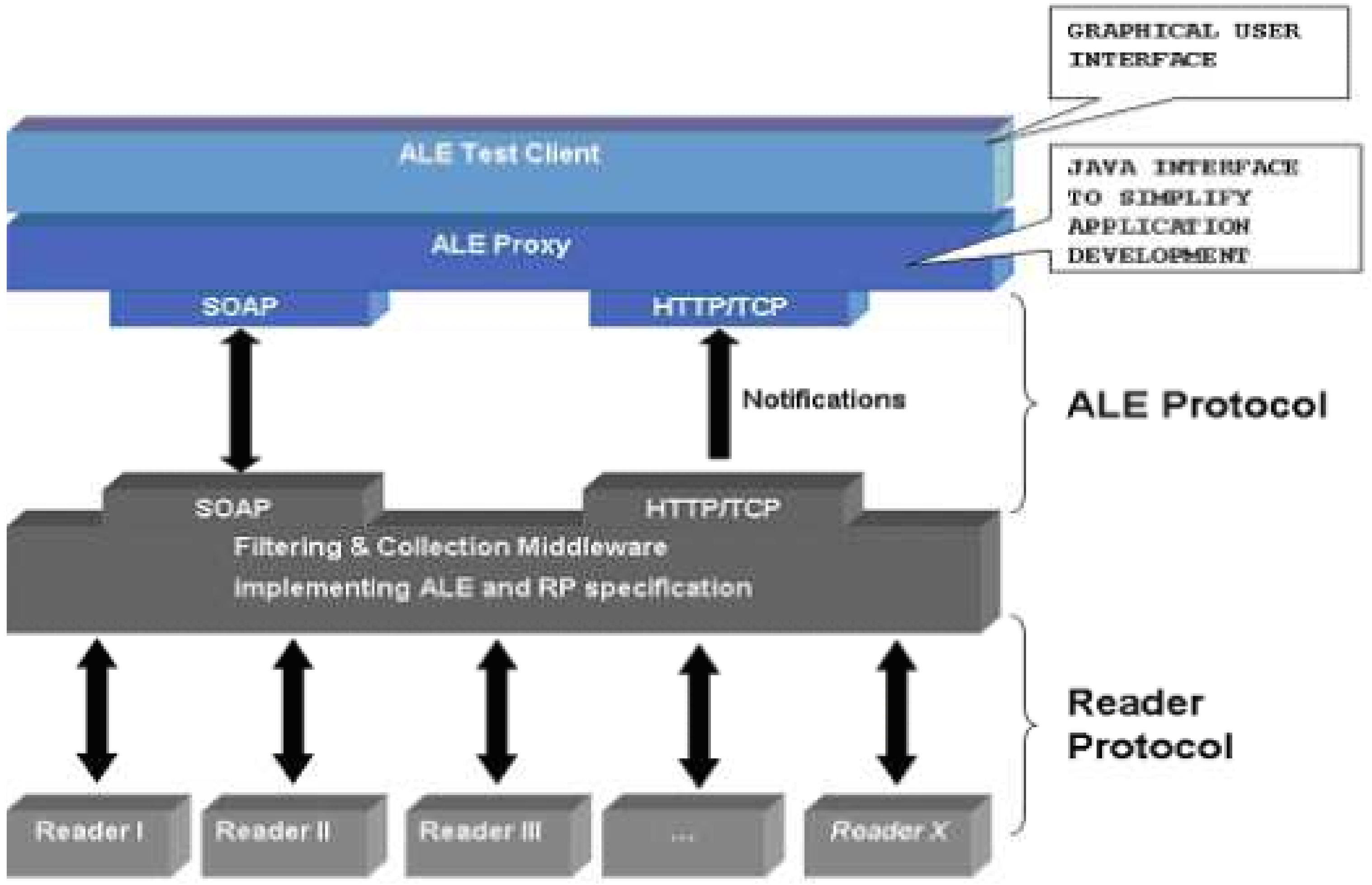
## ECReportGroup



| Field | Type | Description |
|---|---|---|
| groupList | ECReportGroupList | Null if the includeEPC, includeTag, includeRawHex, and includeRawDecimal fields of the corresponding ECReportOutputSpec are all false and the fieldList in the corresponding ECReportOutputSpec is empty (unless ECReportOutputSpec has vendor extensions that cause groupList to be included). Otherwise, an ECReportGroupList instance containing data read from the Tags in this group. |
| groupCount | ECReportGroupCount | Null if the includeCount field of the corresponding ECReportOutputSpec is false (unless ECReportOutputSpec has vendor extensions that cause groupCount to be included). Otherwise, the number of Tags in this group. |

Table 48. ECReportGroup Fields

## Implementing EPC ALEv1.0 implementation

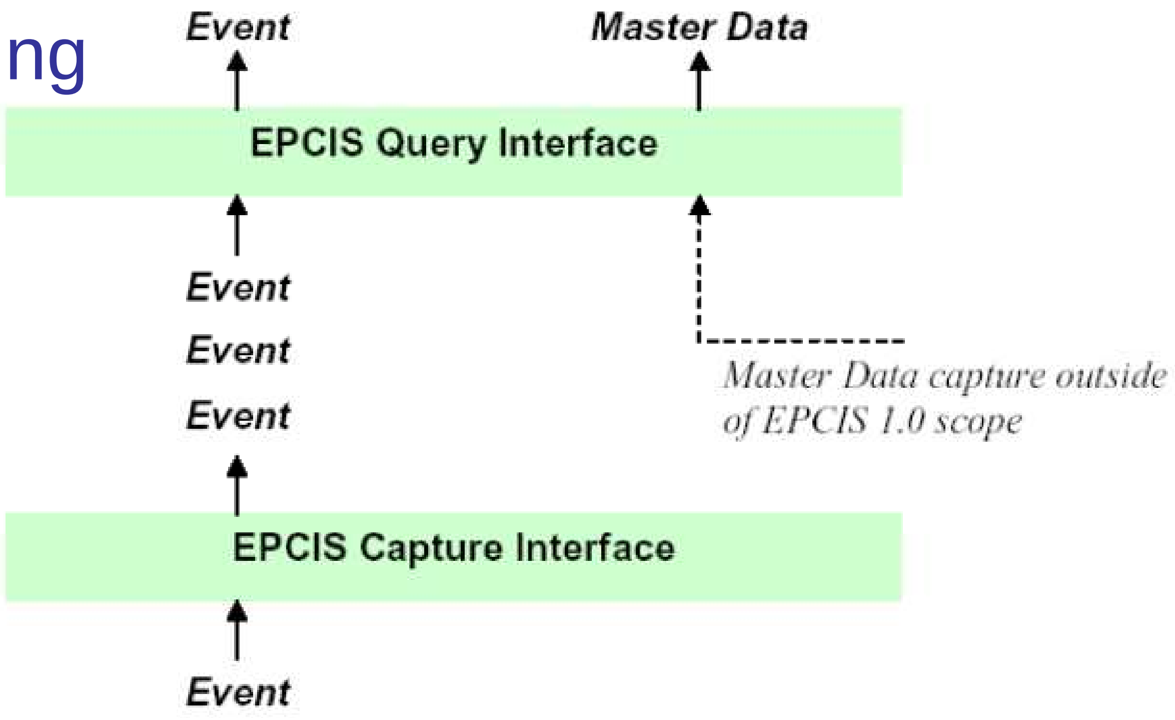


# *EPC-IS, Business Event Generation and Adding the Business Context*

**Athens Information Technology**

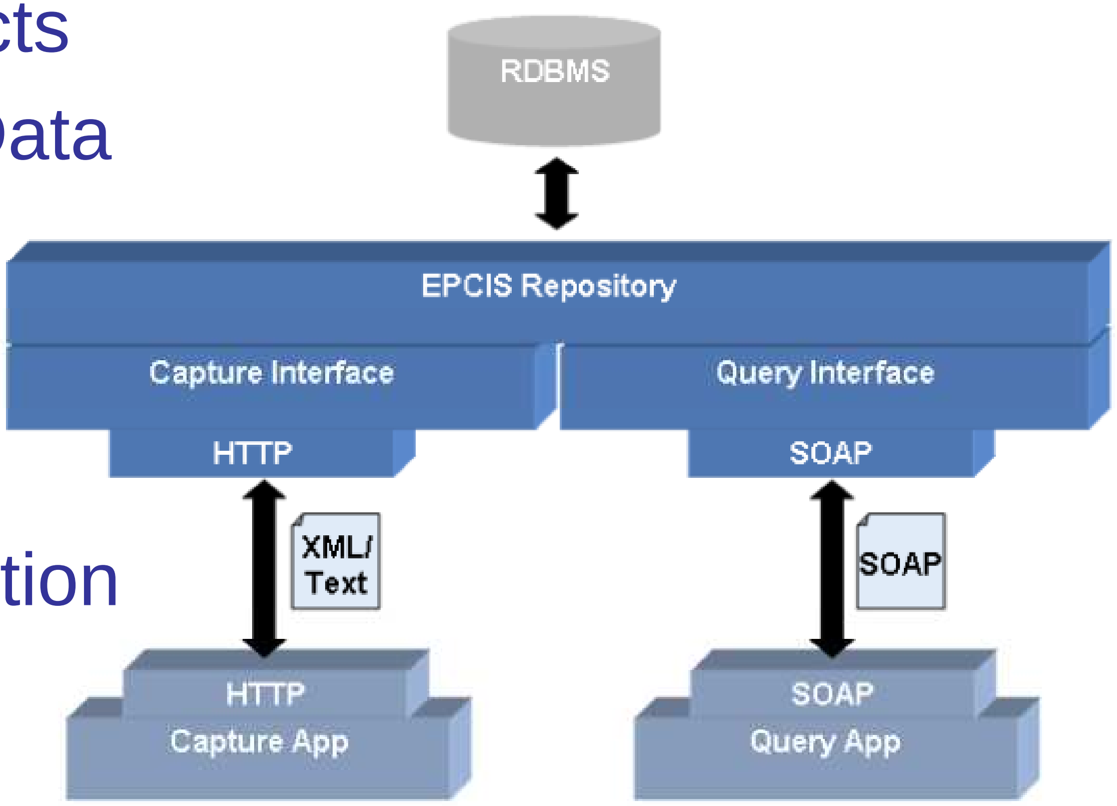
# The EPC ISv1.0 Repository (1)

- Interfaces for
  - Events capturing
  - Events accessing



## The EPC ISv1.0 Repository (2)

- Data Base role
  - Store event objects
  - Provide Master Data
- “Master Data”:
  - Location names
  - Class literals
  - Business transaction types



## EPC Information System (IS) Events (1)

**or what, where, when, and why…**

- EPC IS Event types
  - Object Event
  - Aggregation Event
  - Transaction Event
  - Quantity Event

## EPC Information System (IS) Events (2)

**or what, where, when, and why…**

- EPC IS Semantics
  - EPC (what – retrospective)
  - Parent EPC (what – aggregation)
  - Read Point ID (where – retrospective)
    - A Read Point is a discretely recorded location that is meant to identify the most specific place at which an event took place.

## EPC Information System (IS) Events (3)

**or what, where, when, and why…**

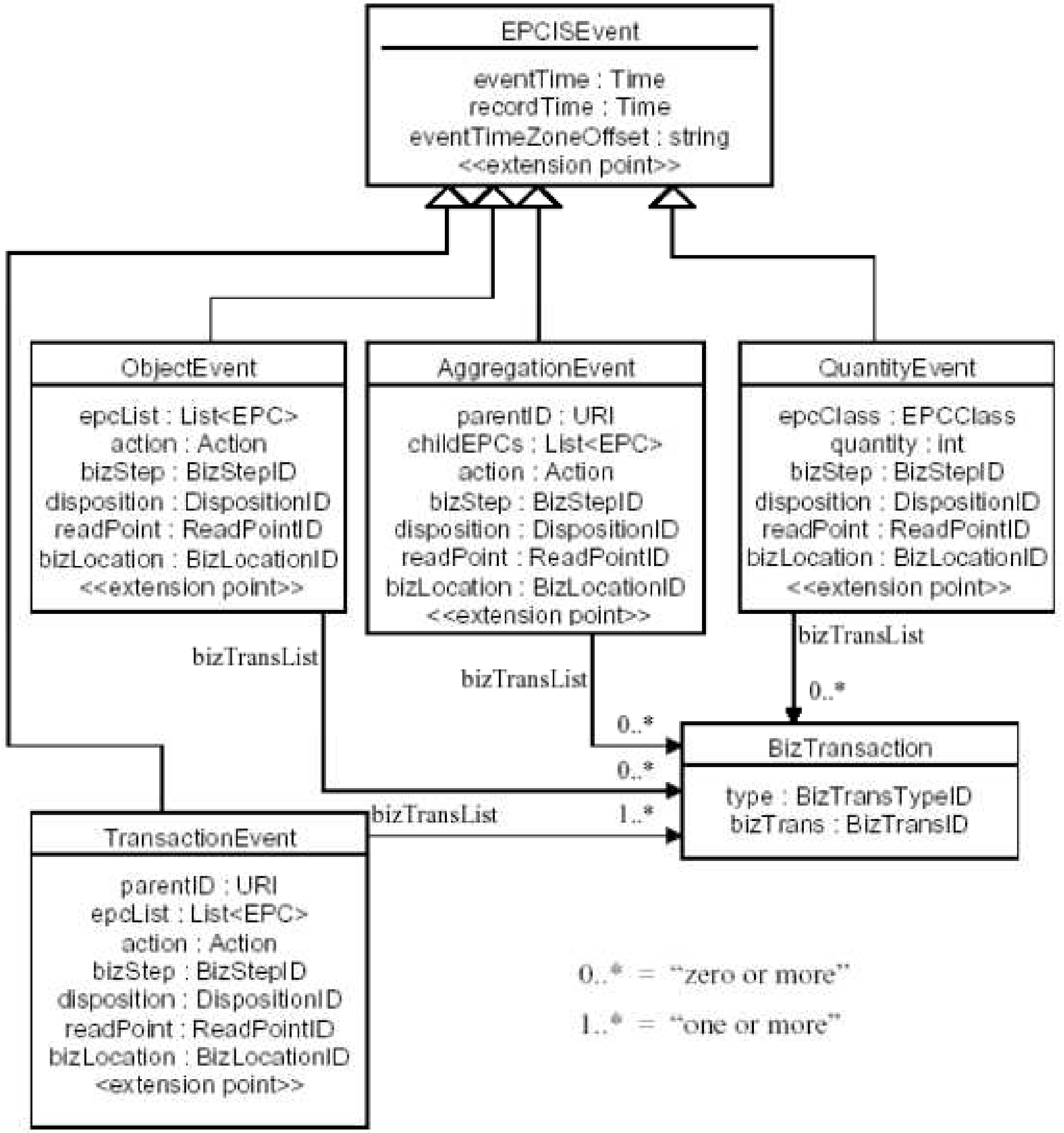
- EPC IS Semantics
  - Business Location ID (where – prospective)
    - A Business Location is a uniquely identified and discretely recorded location that is meant to designate the specific place where an object is assumed to be following an EPCIS event until it is reported to be at a different Business Location by a subsequent event.
  - Business Step ID (why – retrospective)
    - The Business Step of an event specifies its business context (e.g. shipping)

## EPC Information System (IS) Events (4)

**or what, where, when, and why…**

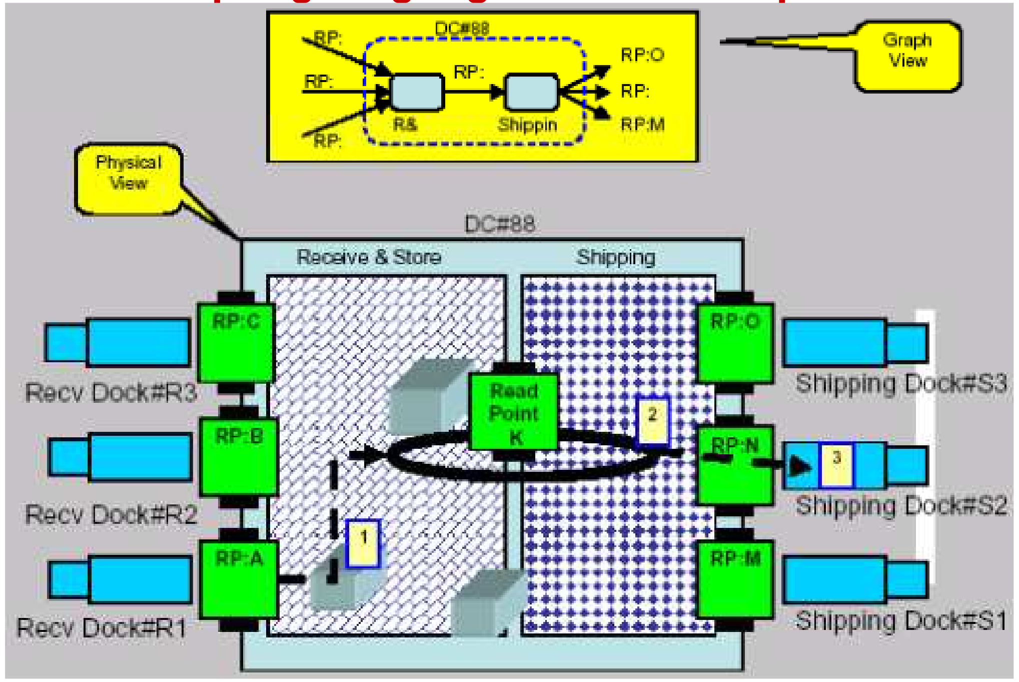
- EPC IS Semantics
  - Disposition ID (why – prospective)
    - The Disposition of an event specifies the business condition of the event’s objects, subsequent to the event
- Event Time (when)

# EPCISEvent


EPCISEvent
eventTime : Time
recordTime : Time
eventTimeZoneOffset : string
<<extension point>>
ObjectEvent
epcList : List<EPC>
action : Action
bizStep : BizStepID
disposition : DispositionID
readPoint : ReadPointID
bizLocation : BizLocationID
<<extension point>>
AggregationEvent
parentID : URI
childEPCs : List<EPC>
action : Action
bizStep : BizStepID
disposition : DispositionID
readPoint : ReadPointID
bizLocation : BizLocationID
<<extension point>>
QuantityEvent
epcClass : EPCClass
quantity : int
bizStep : BizStepID
disposition : DispositionID
readPoint : ReadPointID
bizLocation : BizLocationID
<<extension point>>
bizTransList
0..*
bizTransList
0..*
bizTransList
0..*
BizTransaction
type : BizTransTypeID
bizTrans : BizTransID
bizTransList
1..*
TransactionEvent
parentID : URI
epcList : List<EPC>
action : Action
bizStep : BizStepID
disposition : DispositionID
readPoint : ReadPointID
bizLocation : BizLocationID
<extension point>>
0..* = "zero or more"
1..* = "one or more"

# EPC IS Event sequences

## or the scripting language of business processes…



| EPC | Time | Read Point | Business Location |
|---|---|---|---|
| 123 | 7:00 | RPDC88-A | DC#88.ReceiveStore |
| 123 | 9:00 | RPDC88-K | DC#88.ShippingProduct |
| 123 | 9:30 | RPDC88-N | DC#88.Transit |

## Warehouse Management: Modeling Company & Warehouses (1)

- Modeling the Company for Warehouse Management
  - Logical spaces identified as Warehouses (Wn (n = 0, 1, 2,…))
  - Organized in an hierarchical manner in a way that each warehouse is contained within another warehouse
  - All warehouses can be collectively aggregated under W0, which can be considered as a physical central warehouse or the company itself

## Warehouse Management: Modeling Company & Warehouses (2)

- Child logical warehouses may correspond to physical warehouses or other units of storing capacity down the hierarchy
  - Shelves that are contained within a physical warehouse space

## Warehouse Management: Containers (1)

- Warehouse management processes
  - Based on tagged containers (Cn (n = 0, 1, 2,….,))
  - Typically: pallets, carton boxes, carts, containers, ...
  - Organized in an hierarchical fashion
    - Containers (e.g., pallets) can contain other containers (e.g., carton boxes)

## Warehouse Management: Containers (2)

- Warehouse management processes
  - A container is situated to a parent logical warehouse
  - A container (Cn) is contained in a warehouse, as soon as this warehouse contains a parent container of (Cn)

# Container Vs. Logical Warehouse

- Both containers and logical warehouses can contain other containers and/or items
- Key difference
  - – When items within a container move, the container moves as well, whereas
  - – When items within a logical warehouse move, the logical warehouse does not move
  - – One logical warehouse has typically one parent object (i.e. the parent warehouse)
  - – A container has typically two parent objects (i.e. a parent warehouse and a parent container)

## Examples of Elementary RFID enabled Business Processes for Warehouse Management

- Elementary Warehouse Management Processes
  - Receiving
  - Moving with Warehouses
  - Pick & Pack
  - Order Shipment
  - Inventory
- Each of the above processes is associated with a number of Business Events
  - RFID events comprising business semantics
  - Enhanced EPC-IS events

## Overview of Shipment Process (1)

- Business process takes place in the scope of the “shipping” warehouse ($W_S$)
  - Products that have to be shipped are assembled
  - Moving items and containers (with items) out of warehouse $W_S$
- Assumption
  - Containers have been put within carts during order collection (common)

## Overview of Shipment Process (2)

- During shipment process these aggregations are deleted
  - Items and containers are moved out of the pick & pack carts.
  - New aggregation events signify the creation of packing lists for the shipment process.
  - Objects are moved out of the warehouse

## Overview of Shipment Process (2)

- During shipment process these aggregations are deleted
  - Transaction events are issued to convey and control the status of the process.
    - Transaction observed events provide insight on the objects that have been shipped
    - A Transaction finish event is issued
    - The system can automatically check whether the packing list coincides with the shipment list

## Business Events for Order Shipment (1)

- Aggregation Event denoting that objects are moved out of the cart ($C_n$) (i.e. aggregation deleted)

| Event Description | | | |
|---|---|---|---|
| EventType | Time | bizStepID | dispositionID |
| AggregationEvent | Time | Null | Null |
| bizLocationID | readPointID | EPC | parentEPC |
| Null | Null | <EPC List> | $C_n$ |
| Action | bizTransactionTypeID | bizTransactionID | |
| Delete | Null | Null | |

## Business Events for Order Shipment (2)

- Aggregation Event denoting that a whole group of objects ($C_m$) (e.g., package) are moved out of the cart ($C_n$) (i.e. aggregation deleted)

| Event Description | | | |
|---|---|---|---|
| EventType | Time | bizStepID | dispositionID |
| AggregationEvent | Time | Null | null |
| bizLocationID | readPointID | EPC | parentEPC |
| Null | Null | $C_m$ | $C_n$ |
| Action | bizTransactionTypeID | bizTransactionID | |
| Delete | Null | Null | |

## Business Events for Order Shipment (3)

- Packaging of objects within a container ($C_n$)

| Event Description | | | |
|---|---|---|---|
| EventType | Time | bizStepID | dispositionID |
| AggregationEvent | Time | null | null |
| bizLocationID | readPointID | EPC | parentEPC |
| $W_S$ | $W_S$ | <EPC List> | $C_n$ |
| Action | bizTransactionTypeID | bizTransactionID | |
| Add | Null | null | |

## Business Events for Order Shipment (4)

- Objects leaving the Warehouse WS where the shipment is conducted (i.e. Object Event Delete)

| Event Description | | | |
|---|---|---|---|
| EventType | Time | bizStepID | dispositionID |
| ObjectEvent | Time | Null | Null |
| bizLocationID | readPointID | EPC | ParentEPC |
| Null | $W_S$ | <EPC List> | Null |
| Action | bizTransactionTypeID | bizTransactionID | |
| Delete | Null | Null | |

# Business Events for Order Shipment (5)

- Transaction Event for Objects that have been shipped

| Event Description | | | |
|---|---|---|---|
| EventType | Time | bizStepID | dispositionID |
| TransactionEvent | Time | $D_m$ | $D_n$ |
| bizLocationID | readPointID | EPC | parentEPC |
| Null | Null | <EPC List> | Null |
| Action | bizTransactionTypeID | bizTransactionID | |
| Observed | Null | $BT_n$ | |

## Business Events for Order Shipment (6)

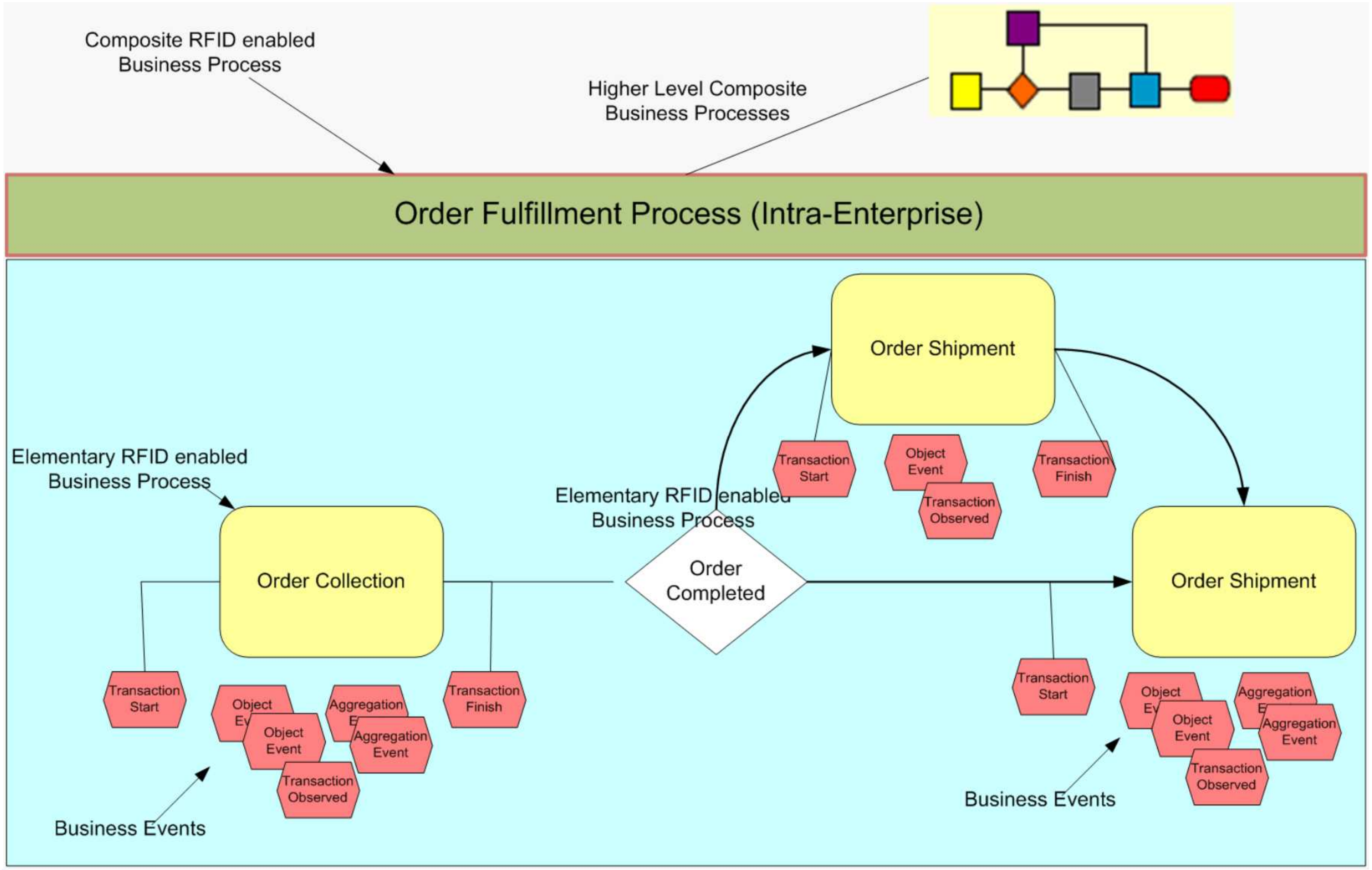
- Transaction event for concluding the order shipment process

| Event Description | | | |
|---|---|---|---|
| EventType | Time | bizStepID | dispositionID |
| TransactionEvent | Time | $D_n$ | Null |
| bizLocationID | readPointID | EPC | parentEPC |
| Null | null | <EPC List> | Null |
| Action | bizTransactionTypeID | bizTransactionID | |
| Delete | Null | $BT_n$ | |

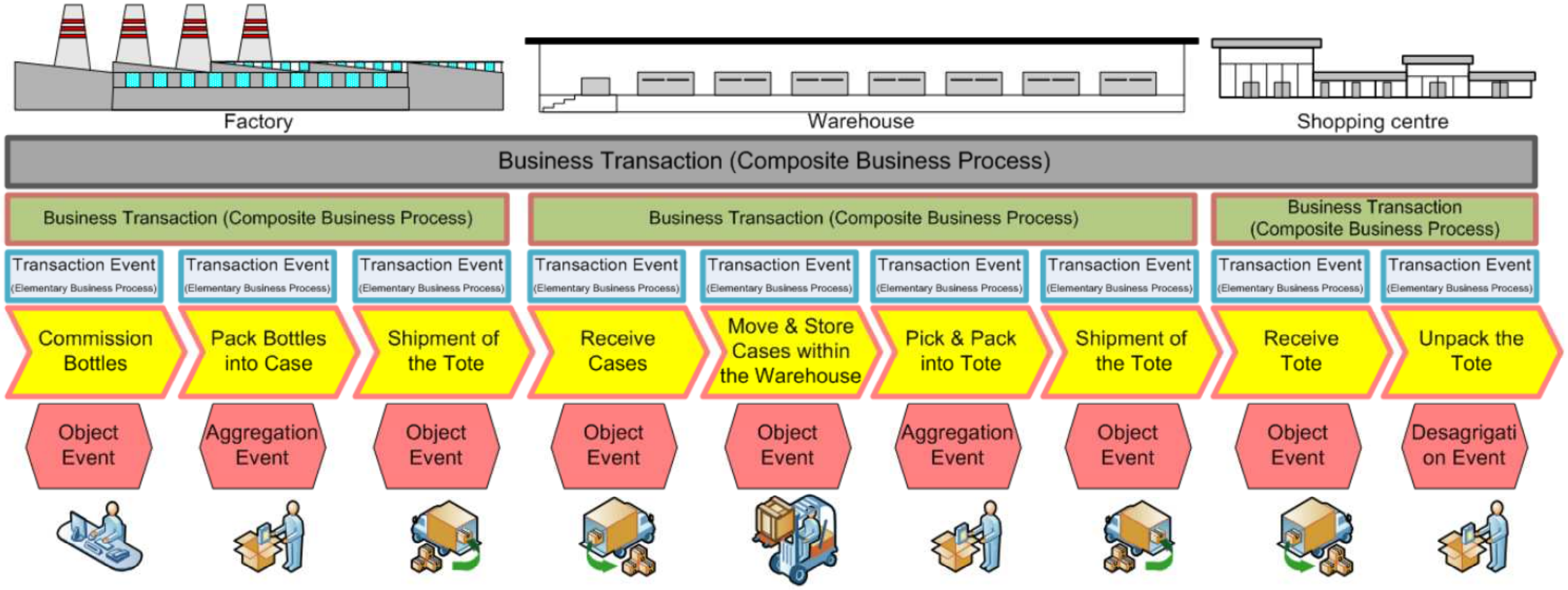
## Composite (RFID-enabled) Business Processes

- Elementary RFID enabled Business Processes can be combined into composite (more complex) processes
  - E.g., If (Receiving == OK) then (Move)
  - E.g., If (Order Collection == OK) then (Order Shipment)
- Hierarchical approach
  - Higher Level (composite) Business Processes, will be assembled by lower-level ones

# Business Process Management Concept



# End-to-End Process Management Concept



## Moving Beyond Warehouse Management: Generality

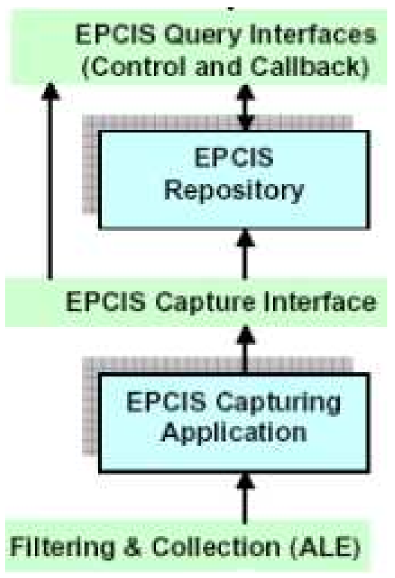
- Is the approach general?
  - Yes: As soon as RFID consultants can describe an elementary process in terms of RFID business events (e.g., EPCIS events)
  - Consortium user experts (e.g., PV, SENSAP) will engage with process descriptions
  - Process Description can be customized for different enterprises and SMEs
    - By RFID Consultants
    - With much less effort than programming
  - Goal: Produce templates for different process & industries

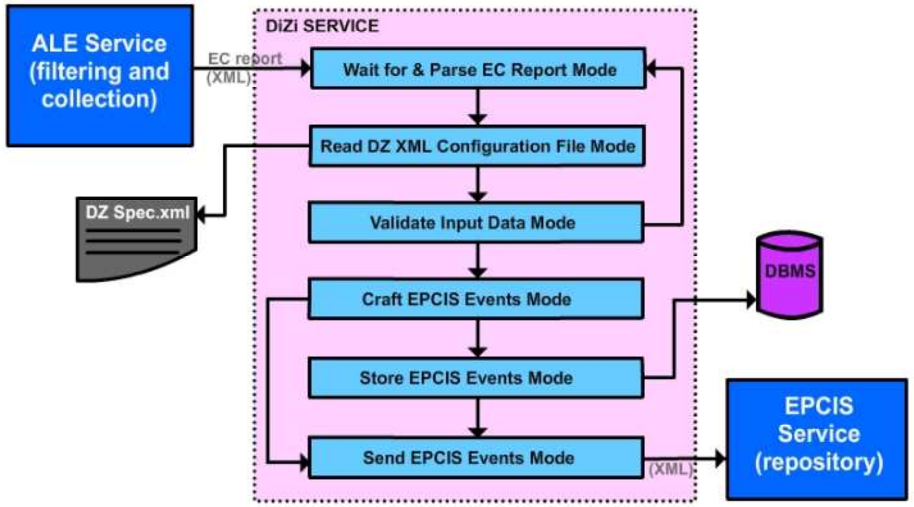
## Moving Beyond Warehouse Management: Scalability

- Is the approach scalable?
  - – Yes: Hierarchically multi-level composite processes can be assembled

# The Business Event Generating (BEG) Engine

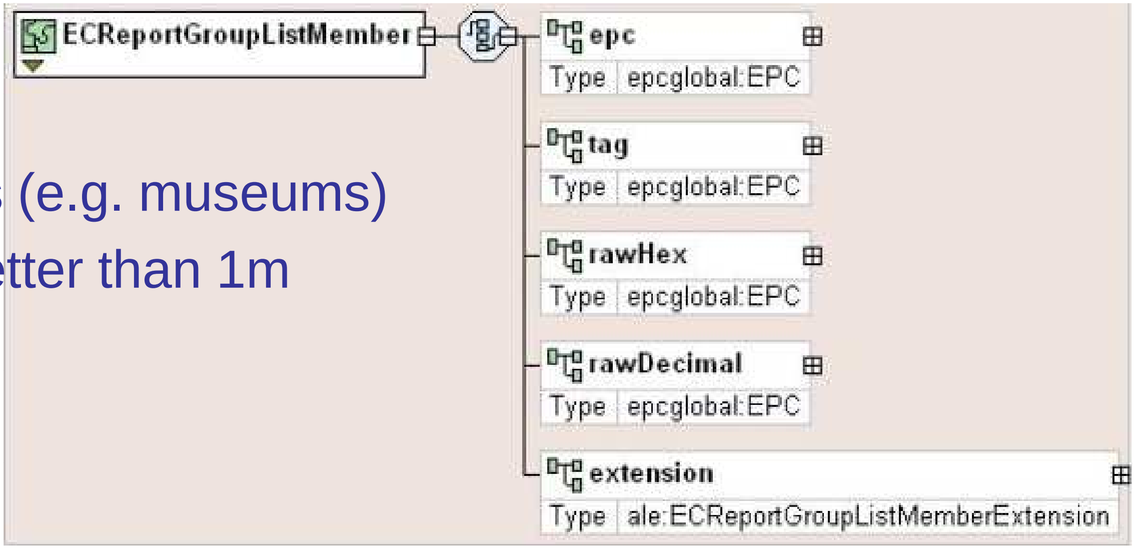
- The role of EPCIS Capturing Application
  - Parse ALE reports
  - Fuse it with business context data
  - Prepare EPCIS compliant events
  - Submit the events to the IS Repository
- BEG Spec: generic configuration description
  - Static or Dynamic submission (adaptable)





## Using BEG algorithms for proximity sensing

- Algorithm
  - Make ALE report Tag occurrences within each Event Cycle
  - Make Reader report Tag occurrences within each Read Cycle
  - Use “majority polling” to estimate the Tag nearest to Reader
  - Reduce Reader transmission power to increase spatial resolution
- Applications
  - Indoor location systems (e.g. museums)
  - Positioning accuracy better than 1m



# References – Additional Reading

- The Application Level Events (ALE) Specification, Version 1.1, EPCglobal, 2008, available online at http://www.epcglobalinc.org/standards/ale
- EPC Information Services (EPCIS) Specification, Version 1.0, EPCglobal, 2007, available online at http://www.epcglobalinc.org/standards/epcis
- EPC Reader Protocol Standard, Version 1.1, EPCglobal 2006, available online at http://www.epcglobalinc.org/standards/rp