Pushing Performance

The Application Level Events (ALE) Vendor Specification
Vendor HARTING IT Software Development GmbH & Co KG
Version 2.1.4

Project: Ha-VIS Middleware
Document Owner: HARTING IT Software Development GmbH & Co KG
Document Date: 2014–09–30
Document State: Approved

Document Version History

Version	State	Author	Date	Changes
2.1.0	Approved	LEB,VAB	2014-01-10	Initial Creation
2.1.1	Approved	LEB	2014-03-31	Revised Edition
2.1.2	Approved	LEB	2014-06-30	Revised Edition
2.1.3	Approved	LEB	2014-09-30	Revised Edition
2.1.4	Approved	LEB	2015-01-10	Revised Edition

Table of Contents

List of Tables

List of Abbreviations

AC	Access Control
ALE	Application Level Events
API	Application Programming Interface
CC	Command Cycle
DNS	Domain Name System
EC	Event Cycle
EPC	Electronic Product Code
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
ISO	International Organization for Standardization
LLRP	Low Level Reader Protocol
LR	Logical Reader
PC	Port Cycle
RC	Reader Configuration
RFC	Request for Comments
RNG	Random Number Generator
RTC	Real Time Clock
SQL	Structured Query Language
TCP	Transmission Control Protocol
TID	Transponder Identifier
TM	Tag Memory
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XML	Extensible Markup Language
XSD	XML Schema Definition

1 Introduction

This document specifies vendor extensions and implementation specific behavior of the ALE HARTING IT Software Development GmbH & Co KG Vendor implementation of the ALE 1.1.1 standard specification.

2 ALE Vendor Interfaces

The HARTING IT Software Development GmbH & Co KG vendor specification for ALE defines two additional interfaces, as defined below.

Interface	Description	Normative section of this document
Digital Input and Output API	An interface through which clients may cause operation to be performed on digital inputs and outputs.	Section 9
Reader Configuration API	An interface through which clients may obtain configuration information from logical readers.	Section 11

Table 1: ALE Vendor APIs

2.1 UML Notation for APIs

See Section 4.1 of the ALE specification document.

2.2 Version Introspection Methods

Each of the two HARTING IT Software Development GmbH & Co KG Vendor APIs includes a pair of methods having the following signature:

```
---
getStandardVersion() :  String
getVendorVersion() :  String
```

This ALE vendor implementation implements these methods for the Vendor APIs as specified in the following table:

Method	Description
`getStandardVersion`	Returns a string that identifies what version of the specification this implementation of the API complies with. The possible values for this string are defined by HARTING IT Software Development GmbH & Co KG. The implementation returns a string corresponding to a version of this specification to which the API implementation fully complies, and returns the string corresponding to the latest version to which it complies. To indicate compliance with this version 1.1 of the ALE specification, the implementation returns the string 1.1.
`getVendorVersion`	Returns a string that identifies what vendor extensions of the API this implementation provides. This implementation returns a non-empty string. The value returned is a URI where this ALE implementation is the owning authority. This URI is a HTTP URL which leads to a copy of this document.

Table 2: Version Interface Methods

For the five standard ALE APIs see section 4.3 of the ALE specification document.

2.3 Classes Common to the Reading, Writing and Digital Input and Output APIs

The following six classes are used by all three the Reading API, the Writing API and the Digital Input and Output API. While their names begin with EC prefix used for Reading API classes, they should be understood as belonging equally to the Reading API, the Writing API and the Digital Input and Output API.

Class	Specified in Section of ALE specification Document
`ECTime`	8.2.2
`ECTimeUnit`	8.2.3
`ECTrigger`	8.2.4
`ECFilterListMember`	8.2.8
`ECReaderStat`	8.3.10

Table 3: List of Common ALE Classes

Class	Specified in Section of this Vendor specification Document
`ECSightingSignalStat`	5.2.3

Table 4: List of Common Vendor Classes

2.4 Scoping of Names

See section 4.6 of the ALE specification document.
The following table enumerates additional namespaces that are implied by this Vendor-Extension.

Namespace	Section	Scope
PCSpec name	9	Global
PCReport	9.2.3	Enclosing PCSpec

Table 5: List of Namespacess

3 ALE Concepts and Principles of Operation

This section describes the concepts and principles of operation that underlie the specification of the ALE Digital Input and Output API. See also section 5 of the ALE specification document.

3.1 Port Cycles

A port cycle is the smallest unit of interaction between an ALE client and an ALE implementation through the ALE Digital Input and Output API. A port cycle is an interval of time during which operations performed on ports. At the conclusion of a port cycle, a report is sent to the ALE client containing information about what events cause the operations and what the results were.

As in an event cycle, the ALE client specifies when a port cycle starts and stops. During the port cycle, the ALE implementation uses one or more readers to operate on ports when a tag falls within the detection zone of a reader or a trigger event occurred.

The interaction between an ALE client and an ALE implementation through the Digital Input and Output API is similar to the description of the Reading API and Writing API from the ALE specification document section 5.2 and section 5.3. Namely,

1. A client provides to the ALE implementation an *port cycle specification* (PCSpec), which specifies
   - one or more readers (this is done indirectly, as explained in Section 10 of the ALE specification document)
   - port cycle boundaries, and
   - a set of reports with operations to apply to ports. Each report includes
     - a filter list that specifies which tags cause port operations to be processed,
     - a trigger list that specifies which trigger events cause port operations to be processed, and

- an ordered list of operations to perform on specific ports

2. The ALE layer responds by carrying out the operations on ports, and returning a report that describes what processing was performed on ports, which event causes the port operations and what the results of the operations are.

3.2 Execution of Event and Command Cycles

See section 5.6 of the ALE specification document.

If the HARTING IT Software Development GmbH & Co KG vendor implementation receives a second `poll` call for a CCSpec for which there is already an outstanding `poll` call, and the second `poll` call specifies the same parameter values as the first, this ALE implementation satisfy the second `poll` call by initiating a new command cycle.

3.3 Execution of Port Cycles

The execution of a port cycle is similar to the description of event cycle and command cycles from the ALE specification document section 5.6.

4 Build-in Fieldnames, Datatypes, and Formats

See Section 6 of the ALE specification document

4.1 Build-in Fieldnames

This section defines the implementation depended behavior for fieldnames that are pre-defined by the ALE specification. The HARTING IT Software Development GmbH & Co KG vendor implementation recognizes each fieldname defined in section 6.1 of the ALE specification document and interprets it as defined in the specification. In addition, this ALE implementation implements the Tag Memory API and recognizes fieldnames defined through that API (see section 7 of the ALE specification document).

4.1.1 Type of Tags

When interacting with a Gen2 tag, the HARTING IT Software Development GmbH & Co KG vendor implementation behaves as specified in the ALE specification document section 6.1. When interacting with any other type of tag, this ALE implementation will not consider that tag and will exclude any tag types others then Gen2 from all reports.

4.1.2 Memory Bank Fieldnames

The HARTING IT Software Development GmbH & Co KG vendor implementation recognizes any memory bank fieldnames that are specified in section 6.1 of the ALE specification document.
This ALE implementation supports also reading or writing to the end of a

memory bank therefore it will not raise an ”operation not possible” condition when an attempt is made to read from or write into the `epcBank`, `tidBank` or `userBank` field.

4.1.3 Variable Fieldnames

This HARTING IT Software Development GmbH & Co KG vendor implementation recognizes any string specified in sub-section 6.1.9.2 of the ALE specification document as a valid variable fieldname. This ALE implementation does not support those variable fieldnames for WRITE, READ, ADD and DELETE operations of the Writing API and for the Reading API, therefore an attempt to do so will always raise an ”operation not possible” condition.

4.2 Build-in Datatypes and Formats

This section defines the implementation depended behavior for datatypes and formats that are pre-defined by the ALE specification. The HARTING IT Software Development GmbH & Co KG vendor implementation recognizes each datatype and format defined in section 6.2 of the ALE specification document and interprets it as defined in the specification.

4.2.1 The `bits` Datatype

The HARTING IT Software Development GmbH & Co KG vendor implementation recognizes the string bits as a valid datatype as specified in section 6.2.3 of the ALE specification document.

When writing a value of type bits, for this implementation the table 22 in sub-section 6.2.3.1 of the ALE specification document is used based on the number of bits in the bits value (M) and the number of bits in the field (N).

The case M<N only requires writing the entire bits value to the field beginning at the field’s leftmost position. This ALE implementation also pads the remaining part of the field with *zero bits.*

5 ALE Reading API

See Section 8 of the ALE specification document.

5.1 ECSpec

See Section 8.2 of the ALE specification document.

Note that the HARTING IT Software Development GmbH & Co KG vendor implementation supports `primaryKeyFields` for all logical readers.This ALE implementation will therefore not raise an `ECSpecValidationException` if the `primaryKeyFields` parameter is specified.

5.1.1 ECTrigger

See Section 8.2.4 of the ALE specification document.

5.1.1.1 Real-time Clock Trigger

The HARTING IT Software Development GmbH & Co KG vendor implementation provides support for real-time clock trigger URIs as defined in section 8.2.4.1 of the ALE specification document.

If the `timezone` parameter within a trigger of this form is omitted this ALE implementation choose the time zone configured in the operating system in which the implementation is running.

5.1.1.2 HTTP Trigger

URIs beginning with the string `urn:havis:ale:trigger:http:` are reserved for triggers specified below. The HARTING IT Software Development GmbH & Co KG vendor implementation provides support for trigger URIs of this form. This ALE implementation conforms to the following specification for all such URIs valid according to the specification below.

A HTTP trigger takes the following form:

`urn:havis:ale:trigger:http:name`

where name is specified below.

Field	Syntax	Meaning
`name`	A (non-empty) string that is accepted by the implementation according to section 4.5 of the ALE specification Document.	The name of a http web address which should be observed by the ALE implementation.

Table 6: HTTP Trigger URI Fields

This ALE implementation interprets a trigger of this form as follows. The trigger is delivered each time the specified web address is called via HTTP protocol using the GET operation. The implementation provides a HTTP Web Service to listening on incoming GET operation calls via HTTP protocol in one of the following forms:

`http://host:port/implementation-defined-URL/trigger-name`
`http://host/implementation-defined-URL/trigger-name`

where

- `host` is the DNS name or IP address of the host where the ALE implementation is listening for incoming calls via HTTP protocol using the GET operation.
- `port` is the TCP port on which the ALE implementation is listening for incoming calls via HTTP protocol using the GET operation. The port and the preceding colon character may be omitted, in which case the port defaults to 80.

- `implementation-defined-URL` is the URL part on which the ALE implementation will register triggers defined using the HTTP trigger syntax.
- `trigger-name` is the name of a trigger defined by the user using the HTTP trigger syntax.

Example (non-normative)

The Condition:
The implementation provides a HTTP Web Service at the following URL:

`http://localhost:8080/ALE/Trigger/`

The following trigger URI denotes a trigger that occurs every time the HTTP Web Address `http://localhost:8080/ALE/Trigger/exampleTrigger` is called via HTTP protocol using the GET operation

`urn:havis:ale:trigger:http:exampleTrigger`

5.1.1.3 Port Trigger

URIs beginning with the string `urn:havis:ale:trigger:port:` are reserved for triggers specified below. The HARTING IT Software Development GmbH & Co KG vendor implementation provides support for trigger URIs of this form. This ALE implementation conforms to the following specification for all such URIs valid according to the specification below.

A port trigger takes one of the three following forms:

```
urn:havis:ale:trigger:port:reader.type
urn:havis:ale:trigger:port:reader.type.id
urn:havis:ale:trigger:port:reader.type.id.state
```

where `reader`, `type`, `id`, and `state` are specified below.

Field	Syntax	Meaning
`reader`	A string corresponding to a `reader` defined through the Logical Reader API.	The name of a logical reader with digital inputs and/or outputs which should be observed
`Type`	One of the indicators 'in' or 'out'.	The type of the port that should be observed. 'in' means an input port. 'out' means an output port.
`Id`	A decimal integer corresponding to a port id.	(Optional)The id of the port that should be observed. If the id is omitted the trigger will be fired on every port id of the specified type.

`State`	One of the decimal integers 0 or 1.	(Optional)The state corresponding to the state of the port that should be observed. 1 means the port is set. 0 means the port is not set. If `state` is omitted the trigger will be fired on every port status change regardless of the state.

Table 7: Port Trigger URI Fields

This ALE implementation interprets a trigger of this form as follows. The trigger is delivered each time the given port of the defined reader changed the status to the state corresponding to `state`. If `state` is omitted the trigger is delivered regardless of the port state. If `id` is omitted the trigger is delivered regardless of the port id.

Example (non-normative) The following trigger URI denotes a trigger that occurs every time a specified input port state is set:

```
urn:havis:ale:trigger:port:ExampleReader.in.1.1
```

The following trigger denotes a trigger that occurs every time a specified input port changed its port state:

```
urn:havis:ale:trigger:port:ExampleReader.in.1
```

The following trigger denotes a trigger that occurs every time a unspecified input port changed its port state:

```
urn:havis:ale:trigger:port:ExampleReader.in
```

5.1.2 ECReportSetSpec

The HARTING IT Software Development GmbH & Co KG vendor implementation interprets an instance of `ECReportSetSpec` as specified in section 8.2.6 of the ALE specification document.

Note that for the first event cycle to completed after the subscribe call for a given subscriber, and for a poll call, this ALE implementation refer "the prior set of tags" to the empty set.

5.1.3 ECStatProfileName

Each valid value of `ECStatProfileName` names as statistics profile that can be included in an `ECReports`

<<Enumerated Type>> ECStatProfileName
TagTimestamps
TagCount
ReaderNames
ReaderSightingSignals
<<extension point>>

5.2 ECReports

See Section 8.3 of the ALE specification document.

5.2.1 ECReaderStat

See section 8.3.10 of the ALE specification document.

The HARTING IT Software Development GmbH & Co KG vendor implementation adds one `ECReaderStat` for each reader to the `statBlocks` parameter list of an `ECTagStat` and the `ECTagStat` is included in an `ECReportGroupListMember` if the `ReaderNames` statistics profile was included in the corresponding `ECReportSpec`.

The `readerName` parameter of the `ECReaderStat` refers to a logical reader name as named in the defining `ECSpec`.

5.2.2 ECTagCountStat

`ECTagCountStat` is a subclass of `ECTagStat`. The HARTING IT Software Development GmbH & Co KG vendor implementation includes one `ECTagCountStat` in an `ECReportGroupListMember` if the `TagCount` statistics profile was included in the corresponding `ECReportSpec`. `ECTagCountStat` includes all of the fields in `ECTagStat`, plus the following additional fields:

ECTagCountStat
count : int

This ALE implementation constructs an `ECTagCountStat` as follows:

Field	Type	Description
profile	ECStatProfileName	This field contains the TagCount value of ECStatProfileName.
statBlock	List <ECReaderStat>	This field contains an empty list.
count	int	This field contains the count how often this tag was seen within this event cycle by any reader contributing to this event cycle.

Table 8: ECTagCountStat Fields

5.2.3 ECSightingSignalStat

ECSightingSignalStat is a subclass of ECSightingStat. The HARTING IT Software Development GmbH & Co KG vendor implementation adds one ECSightingSignalStat for each sighting to the sightings parameter list of an ECReaderStat, an ECReaderStat is added for each reader to statBlocks parameter list of an ECTagStat and the ECTagStat is included in an ECReportGroupListMember if the ReaderSightingSignals statistics profile was included in the corresponding ECReportSpec.
An ECSightingSignalStat contains signal parameter information about a single sighting of a tag by a particular antenna of a particular host.
ECSightingSignalStat includes all of the fields in ECSightingStat, plus the following additional fields:

ECSightingSignalStat
host : String
antenna : int
strength : int

This ALE implementation constructs an ECSightingSignalStat as follows:

Field	Type	Description
host	String	This field contains the host name value for this particular sighting.
Antenna	int	This field contains the antenna id value for this particular sighting.
Strength	int	This field contains the strength value for this particular sighting.

Table 9: ECSightingSignalStat Fields

The readerName parameter of the surrounding ECReaderStat refers to a logical reader name as named in the defining ECSpec.

Note the `ECSightingSignalStat` *is used by all three the Reading API, the Writing API and the Digital Input and Output API. Unless otherwise noted, the interpretation of an* `ECSightingSignalStat` *instance is the same in all three APIs.*

6 ALE Writing API

See Section 9 of the ALE specification document.

6.1 CCSpec

See Section 9.3 of the ALE specification document.

6.1.1 CCOpType

The HARTING IT Software Development GmbH & Co KG vendor implementation recognizes every CCOpType value as specified in section 9.3.5 of the ALE specification document.

6.1.1.1 CHECK Operation

The HARTING IT Software Development GmbH & Co KG vendor implementation recognizes the values `epcBank` and `userBank` as valid fieldspec values for the CHECK `CCOpSpecType` as defined in section 9.3.5.1 of the ALE specification document.

This ALE implementation does not support the ISO 15962 standard therefore the CHECK operation will always result in a `CCOpStatus` of `OP_NOT_POSSIBLE_ERROR`.

6.1.1.2 INITIALIZE Operation

The HARTING IT Software Development GmbH & Co KG vendor implementation recognizes the values `epcBank` and `userBank` as valid fieldspec values for the INITIALIZE `CCOpSpecType` as defined in section 9.3.5.2 of the ALE specification document.

This ALE implementation does not support the ISO 15962 standard therefore the INITIALIZE operation will raise an ”operation not possible” condition and this always results in a `CCOpStatus` of `OP_NOT_POSSIBLE_ERROR`.

6.1.2 CCStatProfileName

Each valid value of `CCStatProfileName` names as statistics profile that can be included in a `CCReports`.

<<Enumerated Type>> CCStatProfileName
TagTimestamps
TagCount
ReaderNames
ReaderSightingSignals
<<extension point>>

6.2 CCReports

See Section 9.4 of the ALE specification document.

6.2.1 CCTagTimestampStat

CCTagTimestampStat is a subclass of CCTagStat. The HARTING IT Software Development GmbH & Co KG vendor implementation includes one CCTagTimestampStat in a CCTagReport if the TagTimestamps statistics profile was included in the corresponding CCCmdSpec. CCTagTimestampStat includes all of the fields in CCTagStat, plus the following additional fields:

CCTagTimestampStat
firstSightingTime : dateTime
lastSightingTime : dateTime

This ALE implementation constructs a CCTagTimestampStat as follows:

Field	Type	Description
profile	CCStatProfileName	This field contains the EventTimestamp value of CCStatProfileName.
statBlock	List <ECReaderStat>	This field contains an empty list.
firstSightingTime	dateTime	This field contains the first time within this command cycle that the tag was seen by any reader contributing to this command cycle.
lastSightingTime	dateTime	This field contains the last time within this command cycle that the tag was seen by any reader contributing to this command cycle.

513 Table 10: CCTagTimestampStat Fields

514 6.2.2 CCTagCountStat

515 CCTagCountStat is a subclass of CCTagStat. The HARTING IT Software
516 Development GmbH & Co KG vendor implementation includes one
517 CCTagCountStat in a CCTagReport if the TagCount statistics profile was
518 included in the corresponding CCCmdSpec. CCTagCountStat includes all of
519 the fields in CCTagStat, plus the following additional fields

CCTagCountStat
count : int

520

521 This ALE implementation constructs a CCTagCountStat as follows:

Field	Type	Description
profile	CCStatProfileName	This field contains the TagCount value of CCStatProfileName.
statBlock	List <ECReaderStat>	This field contains an empty list.
Count	int	This field contains the count how often this tag was seen within this command cycle by any reader contributing to this command cycle.

522 Table 11: CCTagCountStat Fields

6.3 Random Number Generator

This HARTING IT Software Development GmbH & Co KG vendor implementation interprets an RNGSpec as specified in section 9.7.2 of the ALE specification document. This ALE implementation does not define extensions to that class in order to provide additional parameters to control the behavior of a random number generator.

7 ALE Logical Reader API

7.1 API

See section 10.3 of the ALE specification document.

The HARTING IT Software Development GmbH & Co KG vendor implementation does not support to change the definition of a logical reader that is used by an `ECSpec`, `CCSpec` or `PCSpec` that is active at the time the change is requested. Therefore the `update`, `addReaders`, `setReaders`, `removeReaders`, and `setProperties` methods will raise an `InUseException` any time a client calls these methods for a logical reader that is in use by an active `ECSpec`, `CCSpec` or `PCSpec`.

7.2 LRSpec

See section 10.4 of the ALE specification document.

In addition the HARTING IT Software Development GmbH & Co KG vendor implementation supports API-defined Base Reader for logical readers see section 7.3. The implementation will therefore not raise a `ValidationException` if the `isComposite` parameter is false.

7.3 API-defined Base Reader

The HARTING IT Software Development GmbH & Co KG vendor implementation supports a mechanism to define a new base reader using the `define` method of the Logical Reader API. This ALE implementation uses the `properties` parameter of an `LRSpec` to forward information about the communication with a physical reader to the ALE.

Therefore the implementation specifies vendor specific properties and organized these properties in namespaces as specified in the below table.

In the following descriptions the term ”Connector” refers to a unit to execute the communication with a physical reader and to provide an abstract interface to interact with this unit. For each type of physical reader a different ”Connector” unit is used for interaction but every ”Connector” provides the same interface. The term ”Controller” refers to a unit to manage the interaction between the ALE and one ”Connector” unit and thereby to the physical reader.

Property Namespace	Description
	If namespace is omitted this means the common namespace is used which indicates that the property is recognized and handled by the Logical Reader API itself (i.e. tag smoothing parameter see section 7.4).
`Controller.`	The `Controller` namespace indicates that the property is recognized and handled by the ”Controller” unit.
`Connector.`	The `Connector` namespace indicates that the property is recognized and handled by the ”Connector” unit.
`Reader.`	The `Reader` namespace indicates that the property is recognized and handled also by the ”Connector” unit, but this namespace is reserved for specific ”Connector” type properties defined by each ”Connector” type.

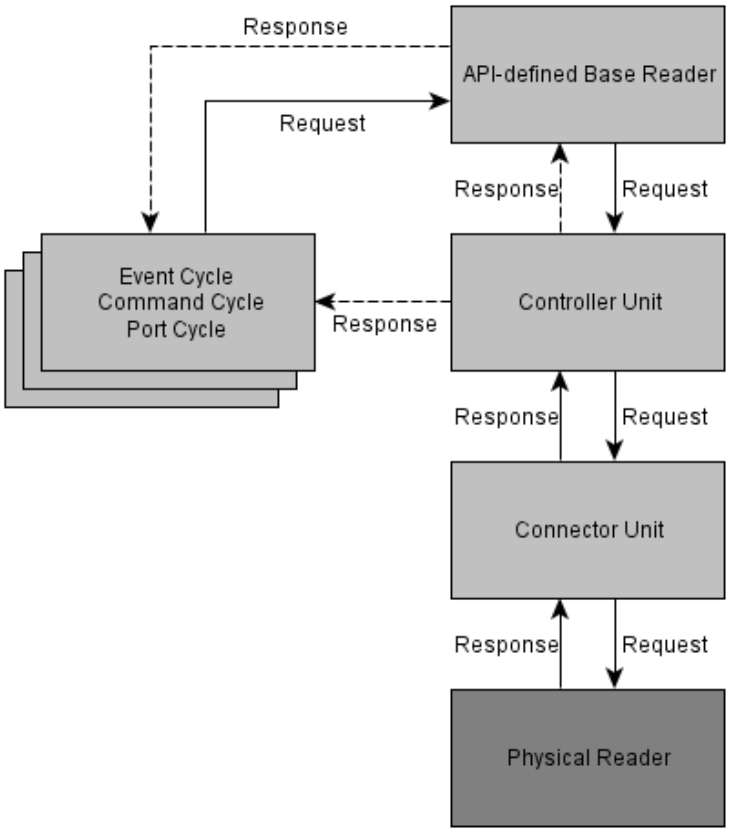
Table 12: Reader Property Namespaces

The `define`, `update` and `setProperties` methods of the Logical Reader API will raise a `ValidationException` under any of the following circumstances:

- A property `name` within the common namespace is not recognized by the implementation.
- A property `name` with specified namespace is within a namespace that is not recognized by the implementation
- The value specified for a property is not legal value for that property.
- The specified `name` is the same as a `name` of another `LRProperty` within the same `LRSpec`.

This ALE implementation will use one ”Controller” unit for every base reader defined using the `define` method of the Logical Reader API and one ”Connector” unit for each ”Controller”.

The following picture shows non-normative the integration of these units in the ALE:



575 Figure 1: API-defined Base Reader Interaction

576 The picture shows the interaction between event, command, port cycle in-
577 stances and an API-defined Base Reader using ”Controller” and ”Connector”
578 units for communication to a physical reader. While the diagram shows two
579 different ways for the ”Controller” unit to deliver a response, the decision if
580 the ”Controller” sends its response directly to the cycle or bypass it over the
581 logical reader is made by the logical reader instance. If the logical reader
582 instance has to handle the response like when tag smoothing is activated
583 (see section 7.4) the response will be passed through the logical reader
584 otherwise the response will be delivered directly.

585 This ALE implementation recognizes the properties for each namespace as
586 defined in the following four sub-sections.

587 7.3.1 Common Namespace

588 The HARTING IT Software Development GmbH & Co KG vendor implementa-
589 tion recognizes a set of `LRProperty` parameters, which an ALE client set us-
590 ing the `properties` parameter of an `LRSpec` or the `setProperties` method
591 of the Logical Reader API for the common namespace. This ALE implemen-
592 tation interprets these parameters as follows:

Property Name	Description
`ReaderType`	A string that refers to the type of ”Connector” unit which is used to interact with the specific physical reader.
`GlimpsedTimeout`	(Optional) A tag smoothing property, see section 7.4 of this document and section 10.6 of the ALE specification document.
`ObservedTimeThreshold`	(Optional) A tag smoothing property, see section 7.4 of this document and section 10.6 of the ALE specification document.
`ObservedCountThreshold`	(Optional) A tag smoothing property, see section 7.4 of this document and section 10.6 of the ALE specification document.
`LostTimeout`	(Optional) A tag smoothing property, see section 7.4 of this document and section 10.6 of the ALE specification document.
`AntennaID`	(Optional) An antenna restriction property, see section 7.5.

Table 13: Common Namespace Properties

The `define`, `update` and `setProperties` methods of the Logical Reader API will raise a `ValidationException` under any of the following circumstances:

- If the `isComposite` parameter within the `LRSpec` is true and `ReaderType` property is specified.
- If the `isComposite` parameter within the `LRSpec` is false and `ReaderType` property is null, omitted or has a value that is not known to the implementation.

7.3.2 Controller Namespace

The HARTING IT Software Development GmbH & Co KG vendor implementation recognizes a set of `LRProperty` parameters, which an ALE client set using the properties parameter of an `LRSpec` or the `setProperties` method of the Logical Reader API for the controller namespace. This ALE implementation interprets these parameters as follows:

Property Name	Description
`Controller.Timeout`	(Optional) A timespan, in milliseconds that governs the timespan after which no response from a ”Connector” unit results in a timeout. Note that a too-small value for `Controller.Timeout` including *zero* value will cause the ”Controller” to achieve the timeout condition immediately

Table 14: Controller Namespace Properties

The `define`, `update` and `setProperties` methods of the Logical Reader API will raise a `ValidationException` under any of the following circumstances:

- If the `isComposite` parameter within the `LRSpec` is true and any property of the controller namespace is specified.
- If the value of `Controller.Timeout` property is a non-null string that is not parseable as a non-negative decimal numeral.

7.3.3 Connector Namespace

The HARTING IT Software Development GmbH & Co KG vendor implementation recognizes a set of `LRProperty` parameters, which an ALE client set using the `properties` parameter of an `LRSpec` or the `setProperties` method of the Logical Reader API for the connector namespace. This ALE implementation interprets these parameters as follows:

Property Name	Description
`Connector.ConnectionType`	A string that refers to the type of connection used by the ”Connector” unit to interact with the physical reader (i.e. TCP).
`Connector.Host`	(Optional) A string that refers to the DNS name or IP address of the physical reader.
`Connector.IP`	(Deprecated) A string that refers to the IP address of the physical reader. Note this property is deprecated, clients should use `Connector.Host` property instead.
`Connector.Port`	(Optional) An unsigned integer that refers to the TCP or COMM port of the physical reader.
`Connector.DeviceID`	(Optional) An unsigned integer that refers to the device ID of the physical reader for USB connection. If `Connector.DeviceID` property is zero the connector will select randomly one available USB reader.
`Connector.Timeout`	(Optional) A timespan, in milliseconds that governs the timespan after which no response from a physical reader results in a timeout. Note that a too-small value for `Connector.Timeout` including zero value will cause the ”Connector” to achieve the timeout condition immediately.

Table 15: Connector Namespace Properties

7.3.3.1 ConnectionType

`ConnectionType` is an enumerated property denoting what type of connections is used for communication with a physical reader.

<<Enumerated Type>> ConnectionType	
TCP	//Indicates a communication via network using TCP protocol
USB	//Indicates a communication via USB port
COMM	//Indicates a communication via COMM port

The `define, update` and `setProperties` methods of the Logical Reader API will raise a `ValidationException` under any of the following circumstances:

- If the `isComposite` parameter within the `LRSpec` is true and any property of the connector namespace is specified.
- If the `Connector.ConnectionType` property is null, omitted or has a value that is not known to the implementation.
- If the value of `Connector.ConnectionType` property is TCP and both `Connector.Host` and `Connector.IP` property are omitted or `Connector.Port` property is omitted.
- If the value of `Connector.ConnectionType` property is USB and `Connector.DeviceID` property is omitted.
- If the value of `Connector.ConnectionType` property is COMM and `Connector.Port` is omitted.
- If the value of `Connector.Host` property is a non-null string that is not valid DNS name nor parseable as an IP-Address of the form [1-255].[1-255].[1-255].[1-255].
- If the value of `Connector.IP` property is a non-null string that is not parseable as an IP-Address of the form [1-255].[1-255].[1-255].[1-255].
- If the value of `Connector.Port` property is a non-null string that is not parseable as a non-negative decimal numeral.
- If the value of `Connector.Timeout` property is a non-null string that is not parseable as a non-negative decimal numeral.

Besides the default properties within the connector namespace each ”Connector” implementation can define a set with additional properties in the connector namespace that are only interpret by this specific ”Connector”. The following sub section will describe these additional properties for the given reader connector implementations.

7.3.3.2 LLRP Connector

The HARTING IT Software Development GmbH & Co KG vendor implementation provides support for physical readers that support the Low Level Reader Protocol (LLRP) for the communication between clients and the reader. The ALE implementation provides an LLRP connector implementation to handle the communication between the ALE and these readers based on LLRP. This connector type can be initialized, with the `LRProperty` parameter `ReaderType` with the value `LLRP`, using the `properties` parameter of an `LRSpec` or the `setProperties` method of the Logical Reader API.

LLRP Type	`ReaderType` Property Value	Description
`LLRP`	”LLRP”	Operate on a physical reader of type LLRP within the Host mode.
`FX7400`	”FX7400”	Operate on a physical reader of type FX7400 within the Host mode.
`FX9500`	”FX9500”	Operate on a physical reader of type FX9500 within the Host mode.

Table 16: LLRP Connector Types

The LLRP Connector recognizes an additional set of `LRProperty` parameters within the connector namespace, and interprets these parameters as follows:

Property Name	Description
`Connector.Keepalive`	(Optional) A timespan, in milliseconds that governs the timespan after which no communication between the physical reader and the connector will cause the connector to send a keepalive message. Note the default value for `Connector.Keepalive` is 30000 milliseconds.
`Connector.InventoryAttempts`	(Optional) An unsigned integer that specifies the amount of inventories that are performed to acquire a specific tag to execute operation, defined trough CC, on. Note the default value for `Connector.InventoryAttempts` is 3.

Table 17: LLRP Connector Properties

7.3.3.3 RF-R Connector

The HARTING IT Software Development GmbH & Co KG vendor implementation provides support for all physical readers of the type HARTING RF-R. The ALE implementation provides connector implementations to handle the communication between the ALE and all supported RF-R readers. These connector types can be initialized according to the following table, using the properties parameter of an `LRSpec` or the `setProperties` method of the Logical Reader API.

RF-R Type	`ReaderType` **Property Value**	Description
`RF-R200`	”RF-R200”	Operate on a physical reader of type RF-R200 within the Host mode.
`RF-R500`	”RF-R500”	Operate on a physical reader of type RF-R500 within the Host mode.

677 Table 18: RF-R Connector Types

678 The RF-R Connector recognizes an additional set of `LRProperty` parame-
679 ters within the connector namespace, and interprets these parameters as
680 follows:

Property Name	Description
Connector.Inventory.Antennas	(Optional) A byte value that specifies which antenna ports are used for inventory attempts for this physical reader. Value = 1 → Antenna No 1 Value = 2 → Antenna No 2 Value = 4 → Antenna No 3 Value = 8 → Antenna No 4 Also every combination of these values by addition of single values is possible. For example: Value = 9 → Antenna No 1, 4
Connector.InventoryAttempts	(Optional) An unsigned integer that specifies the amount of inventories that are performed to acquire a specific tag to execute operation, defined trough CC, on. Note the default value for Connector.InventoryAttempts is 3.
Connector.TagsInField	(Optional) An unsigned integer that specifies the maximum quantity of tags within the reader field at the same time. Note the default value for Connector.TagsInField is 128.
Connector.BlockSize	(Optional) An unsigned integer that specifies the byte size of a block within a tag. Note the default value for Connector.BlockSize is 2.
Connector.BlockCount	(Optional) An unsigned integer that specifies the number of blocks within a tag. Note the default value for Connector.BlockCount is 256.
Connector.ReaderErrorCount	(Optional) An unsigned integer that specifies the amount of inventories without reader error before the error is logged again. Note the default value for Connector.ReaderErrorCount is 3.
Connector.IsoErrorCount	(Optional) An unsigned integer that specifies the amount of inventories without iso error before the error is logged again. Note the default value for Connector.IsoErrorCount is 3.
Connector.AntennaErrorCount	(Optional) An unsigned integer that specifies the amount of inventories without antenna error before the error is logged again. Note the default value for Connector.AntennaErrorCount is 3.

681 Table 19: RF-R500 Connector Properties

7.3.4 Reader Namespace

The HARTING IT Software Development GmbH & Co KG vendor implementation recognizes a set of `LRProperty` parameters, which an ALE client set using the `properties` parameter of an `LRSpec` or the `setProperties` method of the Logical Reader API for the reader namespace.

Properties within the reader namespace are ”Reader” specific properties. Therefore each ”Reader” type defines its own set of properties recognized within the reader namespace. These properties are used to configure the physical reader, each connector provides all configuration properties that are supported by the physical reader the specific connector was implemented for.

A complete set of the properties within the reader namespace could be obtained by a client using the Reader Configuration API (see section 11).

The `define`, `update` and `setProperties` methods of the Logical Reader API will raise a `ValidationException` under any of the following circumstances:

- If the `isComposite` parameter within the `LRSpec` is true and any property of the reader namespace is specified.
- If the value of any property is a non-null string that is not parseable into the expected type.

7.4 Tag Smoothing

The HARTING IT Software Development GmbH & Co KG vendor implementation does fully support tag smoothing as specified in section 10.6 of the ALE specification document for EC, CC and the PC API. Therefore this ALE implementation will not raise a `ValidationException` when a client sets the tag smoothing properties.

7.5 Antenna Restriction

Antenna restriction is a mechanism whereby a composite reader can be configured to use only a single antenna from a single base reader within its readers list to acquire tags. Thereby a logical reader will consider at any point in time a tag to be within view if the tag was read on the specified antenna of the base reader. Only the acquiring of tags underlies these restriction, consequently after a command cycle acquired a matching tag it will operated on this tag regardless which antenna the physical reader uses to execute the operation. Antenna restriction is based upon one parameter, which an ALE client set using the `properties` parameter of an `LRSpec` or the `setProperties` method of the Logical Reader API, as specified in section 10.3 of the ALE specification document. This ALE implementation interprets this parameter as follows:

Property Name	Description
`AntennaID`	An unsigned integer that specifies the id if the antenna to use for acquiring tags trough the logical reader. If a tag is in view of the specified antenna of its reader within the readers list, the logical reader will forward this tag to its observers (event cycle, command cycle, port cycle or another composite reader). Note that an `AntennaID` of *zero* or greater value than the number of antennas of the physical reader will result in the behavior that the reader will never forward a tag because no tag will ever match the requirements set by `AntennaID`.

Table 20: Antenna Restriction Properties

If the property is set to null for a given logical reader the implementation will not use antenna restriction for that logical reader. The `define`, `update` and `setProperties` methods of the Logical Reader API will raise a `ValidationException` under any of the following circumstances:

- If the `isComposite` parameter within the `LRSpec` is false and the `AntennaID` property is specified.
- If the `isComposite` parameter within the `LRSpec` is true, the `AntennaID` property is specified and the reader list contains more than one base reader.
- If the `isComposite` parameter within the `LRSpec` is true, the `AntennaID` property is specified and the `readers` list contains a composite reader.
- If the value of the `AntennaID` property is a non-null string that is not parseable as a non-negative decimal integer numeral.

8 Access Control API

The HARTING IT Software Development GmbH & Co KG vendor implementation does not support the ALE Access Control API specified in section 11 of the ALE specification document.

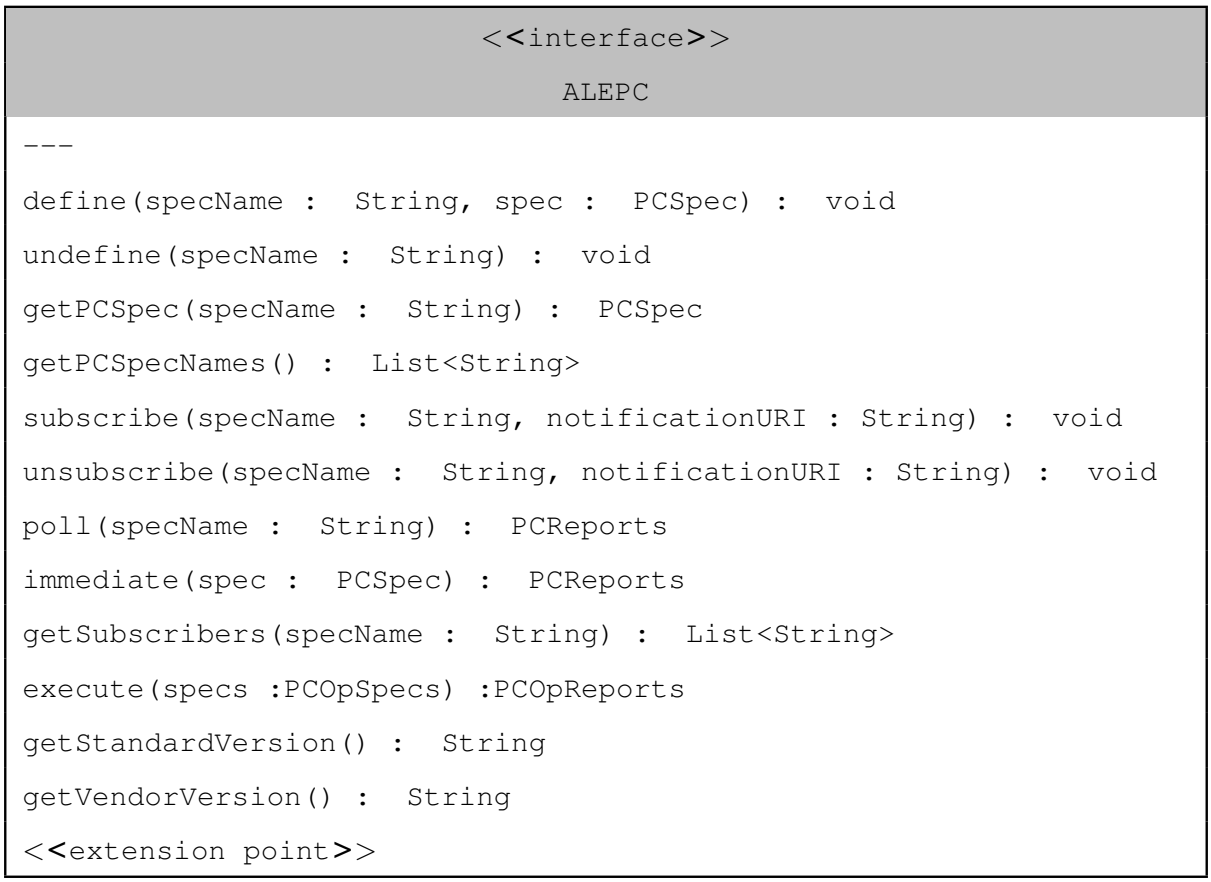
9 ALE Digital Input and Output API

This section defines normatively the ALE Digital Input and Output API. The external interface is defined by the ALEPC interface (Section 9.1). This interface makes use of a number of complex data types that are documented in the sections following section 9.1. The specification of the Digital Input and Output API follows the general rules given in Section 4 of the ALE specification document.

Through the ALEPC interface defined in section 9.1, clients may define and manage port cycle specification (PCSpecs), operate upon ports on-demand by activating PCSpecs synchronously, and enter standing request (subscriptions) for PCSpecs to be activated asynchronously. Results from standing

requests are delivered through the `ALEPCCallback` interface, specified in section 9.4.

9.1 ALEPC - Main API Class



This ALE implementation implements the methods of the ALE Digital Input and Output API as specified in the following table:

Method	Argument /Result	Type	Description
define	specname	String	Creates a new PCSpec having the name specName, according to spec.
	spec	PCSpec	
	[result]	Void	
undefine	specname	String	Removes the PCSpec named specName that was previously created by the define method.
	[result]	Void	
getPCSpec	specname	String	Returns the PCSpec that was provided when PCSpec named specName was created by the define method.
	[result]	PCSpec	
getPCSpecNames	**[result]**	List <String>	Returns an unordered list of the names of all PCSpec that are visible to the caller.
subscribe	specname	String	Adds a subscriber having the specified notificationURI to the set of current subscribers of the PCSpec named specName. The notificationURI parameter both identifies a specific binding of the ALEPCCallback interface and specifies addressing information meaningful to that binding.
	notificationURI	String	
	[result]	Void	
unsubscribe	specname	String	Removes a subscriber having the specified notificationURI from the set of current subscribers of the PCSpec named specName.
	notificationURI	String	
	[result]	Void	
poll	specname	String	Request an activation of the PCSpec named specName, returning the results form the next port cycle to complete.
	[result]	PCReports	
immediate	spec	PCSpec	Creates an unnamed PCSpec according to spec, and immediately requests its activation.
	[result]	PCReports	
getSubscribers	specName	String	Returns an unordered, possibly empty list of the notification URIs corresponding to each of the current subscribers for the PCSpec named specName.
	[result]	List <String>	

execute	specs	PCOpSpecs	Executes a list of PCOpSpec according to specs, and returning the corresponding list of PCOpReport.
	[result]	PCOpReports	
getStandardVersion	**[result]**	String	Returns a string that identifies what version of the specification this implementation of the Digital Input and Output API complies with.
getVendorVersion	**[result]**	String	Returns a string that identifies what vendor extensions this implementation of the Digital Input and Output API provides.

768 Table 21: ALEPC Interface Methods

The primary data type associated with the ALE Digital Input and Output API are the `PCSpec`, which specifies how a port cycle is to be carried out, and the `PCReports`, which contains one or more reports generated from one activation of a `PCSpec`.
`PCReports` instance are both returned from the `poll` and the `immediate` methods, and also sent to subscribers when `PCSpecs` are subscribed to using the `subscribe` method. The next two sections, Section 9.2 and Section 9.3, specify the `PCSpec` and `PCReports` data types in full detail.

9.1.1 Error Conditions

Methods of the ALE Digital Input and Output API signal error conditions to the client by means of exceptions. The following exceptions are defined. All the exception types in the following table are extensions of a common `ALEException` base type, which contains one string element giving the reason for the exception.

Exception Name	Meaning
`SecurityException`	The operation was not permitted due to an access control violation or other security concern.
`DuplicateNameException`	The specified PCSpec name already exists.
`PCSpecValidationException`	The specified PCSpec is invalid. The complete list of rules for generating this exception is specified in Section 4.2.14.
`InvalidURIException`	The URI specified for a subscriber does not conform to URI syntax as specified in [RFC2396], does not name a binding of the `ALEPCCallback` interface recognized by the implementation, or violates syntax or other rules imposed by a particular binding.
`NoSuchNameException`	The specified PSSpec name does not exist.

NoSuchSubscriberException	The specified subscriber does not exist.
DuplicateSubscriberException	The specified PCSpec name and subscriber URI is identical to a previous subscription that was created and not yet unsubscribed.
ImplementationException	A generic exception raised by the implementation for reasons that are implementation-specific. This exception contains one additional element: a severity member whose values are either ERROR or SEVERE. ERROR indicates that the ALE implementation is left in the same state it had before the operation was attempted. SEVERE indicates that the ALE implementation is left in an indeterminate state.

783 Table 22: Exceptions in the ALEPC Interface

784 The exceptions that may be raised by each ALE method are indicated in
785 the table below. This ALE implementation raises the appropriate excep-
786 tion listed below when the corresponding condition described above occurs.
787 If more than one exception condition applies to a given method call, the
788 implementation may raise any of the exceptions that applies.

ALE Method	Exceptions
define	DuplicateNameException PCSpecValidationException SecurityException ImplementationException
undefine	NoSuchNameException SecurityException ImplementationException
getPCSpec	NoSuchNameException SecurityException ImplementationException
getECSpecNames	SecurityException ImplementationException
subscribe	NoSuchNameException InvalidURIException DuplicateSubscriberException SecurityException ImplementationException
unsubscribe	NoSuchNameException NoSuchSubscriberException InvalidURIException SecurityException ImplementationException

poll	NoSuchNameException SecurityException ImplementationException
immediate	PCSpecValidationException SecurityException ImplementationException
getSubscribers	NoSuchNameException SecurityException ImplementationException
execute	PCSpecValidationException SecurityException ImplementationException
getStandardVersion	ImplementationException
getVendorVersion	ImplementationException

Table 23: Exceptions Raised by each ALEPC Interface Method

9.2 PCSpec

A PCSpec is a complex type that describes a port cycle. A port cycle is an interval of time during which ports are operated upon.

A PCSpec contains

(a) one or more logical reader names;

(b) a boundary specification (PCBoundarySpec) that identifies an interval of time;

(c) one or more reports specification (PCReportSpec) that specify operations to be performed on ports of any specified logical readers during the specified interval of time.

The report specification also implies what information is included in a report generated from each port cycle generated from this PCSpec.

PCSpec
logicalReader : List<String>
boundarySpec : PCBoundarySpec
reportSpecs : List<PCReportSpec>
includeSpecInReports : Boolean
<<extension point>>

This ALE implementation interprets the fields of a PCSpec as follows:

Field	Type	Description
`logicalReader`	`List<String>`	An unordered list that specifies one or more logical readers that are used to receive tag events that causes port operations to be processed.
`boundarySpec`	`PCBoundarySpec`	Specifies the starting and stopping conditions for port cycles. See Section 9.2.1
`reportSpecs`	`List <PCReportSpec>`	An ordered list that specifies one or more reports with sequences of operations to apply to ports. See Section 9.2.3
`includeSpecInReports`	`Boolean`	If true, specifies that each `PCReports` instance generated from this `PCSpec` includes a copy of the `PCSpec`. If false, each `PCReports` instance not includes a copy of the `PCSpec`.

Table 24: PCSpec Fields

The `define` and immediate methods raise a `PCSpecValidationException` if any of the following are true for a `PCSpec` instance:

- The `logicalReaders` parameter contains any logical reader names that are not known to the implementation.
- The `boundarySpec` parameter is null or omitted, or the specified `boundarySpec` leads to a `PCSpecValidationException` as specified in Section 9.2.1
- The `reportSpecs` parameter is null, omitted, empty, or any of the members of `reportSpecs` leads to a `PCSpecValidationException` as specified in Section 9.2.3

9.2.1 PCBoundarySpec

A `PCBoundarySpec` specifies how the beginning and end of port cycles are to be determined.

PCBoundarySpec
startTriggerList : List<ECTrigger>
repeatPeriod : ECTime
stopTriggerList : List<ECTrigger>
duration : ECTime
noNewEventsInterval : ECTime
whenDataAvailable : Boolean
<<extension point>>

This ALE implementation interprets the fields of a `PCBoundarySpec` as follows:

Field	Type	Description
`startTriggerList`	`List<ECTrigger>`	(Optional) An unordered list that specifies zero or more triggers that may start a new port cycle for this `PCSpec`.
`repeatPeriod`	`ECTime`	(Optional) Specifies an interval of time for starting a new port cycle for this `PCSpec`, relative to the start of the previous port cycle.
`stopTriggerList`	`List<ECTrigger>`	(Optional) An unordered list that specifies zero or more triggers that may stop a port cycle for this `PCSpec`.
`Duration`	`ECTime`	(Optional) Specifies an interval of time for stopping a port cycle for this `PCSpec`, relative to the start of the port cycle.If omitted or equal to zero, has no effect on the stopping of the port cycle.
`noNewEventsInterval`	`ECTime`	(Optional) Specifies that a port cycle be stopped if no new events are raised within the specified interval. If omitted or equal to zero, has no effect on the stopping of the port cycle.

whenDataAvailable	Boolean	(Optional) If true, specifies that a port cycle be stopped when any event raises that matches the conditions, filter or trigger, of at least one PCReportSpec within this PCSpec. If omitted or false, has no effect on the stopping of the port cycle.

Table 25: PCBoundarySpec Fields

The define and immediate methods raise a PCSpecValidationException if any of the following are true for a PCBoundarySpec instance:

- A negative number is specified for any of the ECTime values duration, repeatPeriod, and noNewEventsInterval.
- Any element of startTriggerList or stopTriggerList does not conform to URI syntax as defined by [RFC2396], or is a URI that is not supported by the ALE implementation. Note that an empty string does not conform to URI syntax as defined by [RFC2396].
- No stopping condition is specified: i.e., stopTriggerList is empty, duration is zero or omitted, noNewEventsInterval is zero or omitted and whenDataAvailable is false.

In the description below, the phrase "if specified" used in reference to repeatPeriod, duration or noNewEventsInterval means that the parameter is specified and is positive (non-zero) number.
The boundarySpec parameter of PCSpec (of type PCBoundarySpec) specifies starting and stopping conditions as referred to in the PCSpec lifecycle specified in Section 3.3.
Within that description, "arrival of a start trigger" means that the ALE implementation receives any of the triggers specified in the startTriggerList for this PCSpec, and "repeat period" means the value if the repeatPeriod parameter, if specified. The phrase "a stopping condition occurred" means the first of the following to occur:

- The duration, when specified, expires (measured from the start of the port cycle).
- When the noNewEventsInterval is specified, no new Events are raised by any reader for the specified interval.
- Any one of the stop triggers specified in the stopTriggerList received.
- The whenDataAvailable parameter is true, and any event occurred that matches the filter or trigger condition of at least one PCReportSpec within this PCSpec. If several matching events are raised in a single reader cycle, the implementation terminate the port cycle after receiving all of those events.

9.2.2 ECTime

See section 8.2.2 of the ALE specification document.

9.2.3 PCReportSpec

A `PCReportSpec` includes (a) a filter specification (`PCFilterSpec`) that has inclusive/exclusive filters to select an amount of tags; (b) a trigger list that defines a list of triggers (`ECTrigger`); (c) on ordered list of one or more operation specifications (`PCOpSpec`), each of which describes a single operation to be performed on a port. During a port cycle, the ALE implementation attempts to carry out the commands specified by the operation specifications on the specified ports.

PCReportSpec
name : String
filterSpec : PCFilterSpec
triggerList: List<ECTrigger>
opSpecs : List<PCOpSpec>
reportIfEmpty : Boolean
statProfileNames : List<PCStatProfileName>
<<extension point>>

This ALE implementation interprets the fields of a `PCReportSpec` as follows:

Field	Type	Description
name	String	Specifies a name for reports generated from this `PCReportSpec`. The ALE implementation copies this name into the `PCReport` instance generated from this `PCReportSpec`.
filterSpec	PCFilterSpec	Specifies how tags are filtered before the event to cause the operations to be processed is raised as specified in Section 9.2.4
triggerList	List<ECTrigger>	Specifies the trigger that can cause the operations to be processed, as specified in Section 5.1.1.
opSpecs	List<PCOpSpec>	An ordered list of `PCOpSpec` instances, each specifying an operation to be carried out on a port, as specified in Section 9.2.5. This ALE implementation will process each event that matches `filterSpec` or `triggerList` acquired during a port cycle.

reportIfEmpty	Boolean	Specifies whether to omit the PCReport instance if the final set of ports is empty, as specified below.
statProfileNames	List <PCStatProfileName>	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 9.2.9.

Table 26: PCReportSpec Fields

The define and immediate methods raise a PCSpecValidationException if any of the following are true for a PCReportSpec instance:

- The specified name is an empty string or is not accepted by the implementation according to Section 4.5 of the ALE specification document.
- The specified name is a duplicate of another report name in the same PCSpec.
- The specified filterSpec leads to a PCSpecValidationException as specified in Section 9.2.4.
- The value of any element of triggerList does not conform to URI syntax as defined by [RFC2396], or is a URI that is not supported by the ALE implementation. Note that an empty string does not conform to URI syntax as defined by [RFC2396].
- Both filterSpec and triggerList are specified for the same PCReportSpec.
- The logicalReaders parameter of PCSpec is null, omitted or an empty list and triggerList is not specified.
- Any element of opSpecs leads to a PCSpecValidationException as specified in Section 9.2.5.
- Any element of statProfileNames is not the name of a known statistic profile.
- If triggerList is specified and any element of statProfileNames refers to a tag statistic which is not evaluable by a trigger event.

Note if both filter and trigger are not specified for a report this is equivalent to an empty filter list where every tag event will match the filter conditions. An empty list will be interpreted by the ALE implementation as not specified.

A PCReports instance includes a PCReport instance corresponding to each PCReportSpec in the governing PCSpec, in the same order specified in the PCSpec, except that a PCReport instance is omitted under the following circumstances:

- If a PCReportSpec has reportIfEmpty set to false, then the corresponding PCReport instance is omitted from the PCReports for this port cycle if the final, filtered set of events is empty.

When the processing of `reportIfEmpty` results in *all* `PCReport` instances being omitted from the `PCReports` for a port cycle, then the delivery of the results to subscribers is suppressed altogether. That is, a result consisting of a `PCReports` having zero contained `PCReport` instances is not sent to a subscriber. This rule only applies to subscribers, a `PCReports` instance always is returned to the caller of `immediate` or `poll` at the end of a port cycle, even if the `PCReports` instance contains zero `PCReport` instances.

The `statProfileName` parameter is a list of `PCStatProfileName`, each of which corresponds to a statistics profile that will be included in the `PCReports`. If the ALE engine does not recognize any name in the list or the combination of event set (trigger or tags) and any statistics profile is not evaluable it raises a `PCSpecValidationException`.

9.2.4 PCFilterSpec

A `PCFilterSpec` specifies what tags will cause port operations to be processed by a `PCReportSpec`.

PCFilterSpec
`filterList : List<ECFilterListMember>`
`<<extension point>>`
`---`

This ALE implementation interprets the fields of a `PCFilterSpec` as follows.

Field	Type	Description
`filterList`	`List <ECFilterListMember>`	Specifies an unordered list of filters, as specified below.

Table 27: PCFilterSpec Fields

The `define` and `immediate` methods will raise a `PCSpecValidationException` if any of the following are true for a `PCFilterSpec`:

- Any element of `filterList` is leads to a `PCSpecValidationException` as specified in Section 8.2.8 of the ALE specification Document.

The `PCFilterSpec` implements a flexible filtering scheme based on a list of `ECFilterListMember` instances (`ECFilterListMember` is shared with the ALE Reading API, and specified in Section 8.2.8 of the ALE specification Document). Each `ECFilterListMember` instances defines a test to be applied to fields of a tag to determine if the tag should cause port operations according to the containing in `PCReportSpec` to be processed. A tag raises an event to cause operations specified in the `PCReportSpec` if it passes the test specified by every `ECFilterListMember` in `filterList`, as defined in Section 8.2.7 and 8.2.8 of the ALE specification Document.

If accessing a field specified by any element of `filterList` causes a ”field

not found” or ”operation not possible” condition, that tag will not be raise an event for this `PCReportSpec`.

9.2.5 PCOpSpec

Each `PCOpSpec` specifies an operation to perform on a port, such as reading a port status or setting a port status. Each `PCOpSpec` has on operation type that specifies which operation to perform and a `portSpec` that indicates which port the operation is processed on. Operations that require input state (such as setting a port status) include the state parameter to specify the input data.

PCOpSpec
opType : PCOpType
portSpec : PCPortSpec
state : Boolean
duration : ECTime
opName : String
<<extension point>>

This ALE implementation interprets the fields of a `PCOpSpec` as follows:

Field	Type	Description
opType	PCOpType	Specifies the operation to be performed.
portSpec	PCPortSpec	Specifies the port to process a port operation on.
State	Boolean	(Conditional) Specifies the `state` source as a boolean value. If `state` is set to true the specified port will be activated. If `state` is set to false the specified port will be deactivated. If `opType` specifies an operation that does not require input state, this parameter must be null or omitted.
Duration	ECTime	(Optional) If `opType` support `duration`, like setting a port status, this parameter specifies the `duration` to set the specified status. If `duration` is null or omitted the value is interpret as infinite. If `opType` does not support `duration` this parameter must be omitted.
opName	String	(Optional)A name for this operation within in `PCOpSpec`. If specified, the value is copied into the corresponding `PCOpReport` instance. If omitted, the `opName` parameter of the corresponding `PCOpReport` instance will be omitted as well.

Table 28: PCOpSpec Fields

The `define, immediate` and `execute` methods raise a `PCSpecValidationException` if any of the following are true for a `PCOpSpec` instance:

- The specified `opType` value is not one of the standard `opType` values specified in section 9.2.6.
- The `portSpec` parameter is null, omitted or invalid according to section 9.2.7.
- The specified `opType` requires a `state`, and `state` is null or omitted.
- The specified `opType` does not require `state`, and `state` is specified.
- The specified `opType` does not require `duration`, and `duration` is specified.
- When `opName` is specified, the specified `opName` is the same as an `opName` of another `PCOpSpec` within the same `PCReportSpec` instance or the list use by the `execute` method.

9.2.6 PCOpType

`PCOpType` is an enumerated type denoting what type of operation is represented by the `PCOpSpec`.

<<Enumerated Type>> PCOpType
READ
WRITE
<<extension point>>

The following table describes each value of `PCOpType`, and interpretation of `portSpec` and `state` within `PCOpSpec` when that `PCOpType` value is specified.

PCOpType Value	**Description**	portSpec	state	duration
READ	Read status from port.	The port to read.	[Must be omitted]	[Must be omitted]
WRITE	Set status of a port.	The port to set.	The value to set for the specified port.	Specifies a timespan to set the port status. If null or omitted sets the status permanently.

Table 29: PCOpType Value

9.2.7 PCPortSpec

A `PCPortSpec` defines a port to execute a port operation on:

PCPortSpec
`id : int`
`reader : String`
`type : PCPortType // INPUT or OUTPUT`
`<<extension point>>`
`---`

This ALE implementation interprets the fields of a `PCPortSpec` as follows:

Field	Type	Description
`id`	`int`	Specifies the port `id`; that is which port to operate upon.
`Reader`	`String`	Specifies the logical reader name; that is which `reader` to operate upon.
`Type`	`PCPortType`	Specified whether this `PCPortSpec` specifies an input or output port. If this parameter is INPUT, the spec defines an input port. IF this parameter is OUTPUT, this spec defines an output port.

Table 30: PCPortSpec Fields

The `define`, `immediate` and `execute` methods raise a `PCSpecValidationException` if any of the following are true for a `PCPortSpec`:

- The `id` parameter is null, omitted or negative number is specified.
- The `reader` parameter is null or omitted or is any logical reader name that is not known to the implementation.
- The specified `type` value is not one of the standard type values specified in Section 9.2.8.
- The specified `reader` is any logical reader name that is not a base reader.

9.2.8 PCPortType

`PCPortType` is an enumerated type denoting what type of port is represented by the `PCPortSpec`.

<<Enumerated Type>> PCPortType
INPUT
OUTPUT
<<extension point>>

9.2.9 PCStatProfileName

Each valid value of `PCStatProfileName` names as statistics profile that can be included in a `PCReports`

<<Enumerated Type>> PCStatProfileName
EventTimestamps
EventCount
ReaderNames
ReaderSightingSignals
<<extension point>>

The `define, immediate` and `execute` methods raise a `PCSpecValidationException` for any of the following circumstances:

- If the `statProfileNames` parameter of the `PCReportSpec` contains `ReaderNames` and the `triggerList` parameter is specified.
- If the `statProfileNames` parameter of the `PCReportSpec` contains `ReaderSightingSignals` and the `triggerList` parameter is specified.

9.2.10 Validation of PCSpecs

The `define, immediate` and `execute` methods of the ALEPC API raises a `PCSpecValidationException` if any of the following are true:

- The specified `specName` is an empty string or is not accepted by the implementation according to Section 4.5 of the ALE specification Document.
- The `logicalReaders` parameter of `PCSpec` contains any logical reader names that are not known to the implementation.
- The `boundarySpec` parameter is null or omitted.
- The `duration, repeatPeriod`, and `noNewEventsInterval` parameter of `PCBoundarySpec` is negative.
- Any element of `startTriggerList` or `stopTriggerList` parameter of `PCBoundarySpec` does not conform to URI syntax as defined by [RFC2396], or is a URI that is not supported by the ALE implementa-

tion. Note that an empty string does not conform to URI syntax as defined by [RFC2396].

- No stopping condition is specified in `PCBoundarySpec`: i.e. `stopTriggerList` is empty, and neither `duration` nor `noNewEventsInterval` nor `whenDataAvailable` is specified.
- Any `PCReportSpec` instance has a name that is an empty string or that is not accepted by the implementation according to Section 4.5 of the ALE specification document.
- Two `PCReportSpec` instances have identical values for their name field.
- The `patList` parameter of any `ECFilterListMember` instance is empty, null, or omitted, or any element of `patList` does not conform to the syntax rules for patterns implied by the specified `fieldSpec`.
- The value of any element of `triggerList` of `PCReportSpec` does not conform to URI syntax as defined by [RFC2396], or is a URI that is not supported by the ALE implementation. Note that an empty string does not conform to URI syntax as defined by [RFC2396].
- Both `filterSpec` and `triggerList` of `PCReportSpec` are defined for the same `PCReportSpec` instance.
- The `logicalReaders` parameter of `PCSpec` is null, omitted or an empty list and `triggerList` parameter of any `PCReportSpec` is not specified.
- The `opType` parameter of a `PCOpSpec` is not one of the standard `opType` values specified in Section 9.2.6.
- The `portSpec` parameter of a `PCOpSpec` is null or omitted.
- The `id` parameter of a `PCPortSpec` is null, omitted or has a negative value
- The `reader` parameter of a `PCPortSpec` is null, omitted or is any logical reader name that is not known to the implementation.
- The `opType` parameter of a `PCPortSpec` is not one of the standard type values specified in Section 9.2.8.
- The `opType` parameter of a `PCPortSpec` requires a `dataSpec`, and `dataSpec` is null or omitted.
- The `opType` parameter of a `PCPortSpec` does not support `dataSpec`, and `dataSpec` is specified.
- The `opType` parameter of a `PCOpspec` does not support `duration`, and `duration` is specified.
- Two or more `PCOpSpec` instances within the same `PCReportSpec` instance or the list used by the `execute` method specify the same (non-empty) `opName`.
- Any value of `PCStatProfileName` is not recognized, or is recognized but the specified statistics report is not supported.
- The `statProfileNames` parameter of the `PCReportSpec` contains `ReaderName` and the `triggerList` parameter is specified.
- The `statProfileNames` parameter of the `PCReportSpec` contains `ReaderSightingSignal` and the `triggerList` parameter is specified.

- The `reader` parameter of a `PCPortSpec` is any logical reader name that is not a base reader.

9.3 PCReports

The `PCReports` object is the output from a port cycle.

PCReports
specName : String
date : dateTime
ALEID : String
totalMilliseconds : long
initiationCondition : PCInitiationCondition
initiationTrigger : ECTrigger
terminationCondition : PCTerminationCondition
terminationTrigger : ECTrigger
pcSpec : PCSpec
reports : List<PCReport>
<<extension point>>

The "meat" of a `PCReports` instance is the ordered list of `PCReport` instances, each corresponding to a `PCReportSpec` instance in the port cycle's `PCSpec`, and appearing in the order corresponding to the `PCSpec`. In addition to the reports themselves, `PCReports` contains a number of "header" fields that provide useful information about the port cycle. The implementation includes these fields according to the following definitions:

Field	Description
specName	The name of the `PCSpec` that controlled this port cycle. In the case of a `PCSpec` that was requested using the immediate method, this name is one chosen by the ALE implementation.
Date	A representation of the date and time when the port cycle ended. For bindings in which this field is represented textually, an ISO-8601 compliant representation is 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 port cycle to the end of the port cycle.

initiationCondition	Indicates what kind of event caused the port 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 PCSpec. These correspond to the possible ways of specifying the start of a port cycle as defined in Section 9.2.1.
initiationTrigger	If initiationCondition is TRIGGER, the ECTrigger instance corresponding to the trigger that initiated the port cycle; omitted otherwise.
terminationCondition	Indicates what kind of event caused the port cycle to terminate: the receipt of an explicit stop trigger, the expiration of the port cycle duration or no events occurred for the prescribed amount of time. These correspond to the possible ways of specifying the end of a port cycle as defined in Section 9.2.1.
terminationTrigger	If terminationCondition is TRIGGER, the ECTrigger instance corresponding to the trigger that terminated the port cycle; omitted otherwise.
PCSpec	A copy of the PCSpec that generated this PCReports instance. Only included if the PCSpec has includeSpecInReports set to true.
Reports	A List containing a PCReport for each PCReportSpec in the corresponding PCSpec. See Section 9.3.3.

Table 31: PCReports Fields

9.3.1 PCInitiationCondition

PCInitiationCondition is an enumerated type that describes how a port cycle was started.

<<Enumerated Type>> PCInitiationCondition
TRIGGER
REPEAT_PERIOD
REQUESTED
UNDEFINE
<<extension point>>

This ALE implementation set the initiationCondition field of a PCReports instance generated at the conclusion of a port cycle according to the condition that caused the port cycle to start, as specified in the following table:

PCInitiationCondition	Event causing the port cycle to start
TRIGGER	One of the trigger specified in startTriggerList of PCBoundarySpec was received.
REPEAT_PERIOD	The repeatPeriod specified in the PCBoundarySpec expired, or the port cycle started immediately after the previous port cycle ended because neither a start trigger nor a repeat period was specified.
REQUESTED	The PCSpec transitioned from the unrequested state to the requested state and startTriggerList in PCBoundarySpec was empty.
UNDEFINE	Used when an outstanding poll call is terminated due to an undefined call, while the PCSpec was in the requested state.

1127 Table 32: PCInitiationCondition Values

1128 Each row of this table corresponds to one of the possible start conditions
1129 specified in 9.2.1.

1130 ### 9.3.2 PCTerminationCondition

1131 PCTerminationCondition is an enumerated type that describes how a port
1132 cycle was ended.

<<Enumerated Type>> PCTerminationCondition
1133 TRIGGER
1134 DURATION
1135 NO_NEW_EVENTS
1136 DATA_AVAILABLE
1137 UNREQUEST
1138 UNDEFINE
1139 <<extension point>>

1140 This ALE implementation set the terminationCondition field of a
1141 PCReports instance generated at the conclusion of a port cycle according to
1142 the condition that caused the port cycle to end, as specified in the following
1143 table:

PCTerminationCondition	Event causing the port cycle to end
TRIGGER	One of the trigger specified in stopTriggerList of PCBoundarySpec was received.
DURATION	The duration specified in the PCBoundarySpec expired.
NO_NEW_EVENTS	No new events were raised within the noNewEventsInterval specified in the PCBoundarySpec.
DATA_AVAILABLE	The whenDataAvailable parameter of the PCBoundarySpec was true and a Port was processed.
UNREQUESTED	The PCSpec transitioned to the unrequested state to the requested state. By definition, this value cannot actually appear in a PCReports instance sent to any client.
UNDEFINE	The PCSpec was removed by an undefined call while in the requested or active state.

1144 Table 33: PCTerminationCondition Values

1145 Each row of this table corresponds to one of the possible stop conditions
1146 specified in 9.2.1.

1147 9.3.3 PCReport

1148 Each PCReportSpec in the PCSpec is associated with a PCReport.

PCReport
1149 reportName : String
1150 eventReports : List<PCEventReport>
1151 <<extension point>>
1152 ---

1153 This ALE implementation constructs a PCReport as follows:

Field	Type	Description
reportName	String	A copy of the reportName field from the corresponding PCReportSpec within the PCSpec that controlled this port cycle.
eventReport	List <PCEventReport>	An unordered list of PCEventReport instances, one for each event occurred during the port cycle that matches the filter or trigger conditions of the corresponding PCReportSpec.

Table 34: PCReport Fields

9.3.4 PCEventReport

A PCEventReport describes what happened during the processing of a single event that cause operations on a port to be processed.

PCEventReport
id : String
opReports : List<PCOpReport>
stats : List<PCEventStat>
<<extension point>>

This ALE implementation constructs a PCEventReport from operation processed on a single Port, as follows:

Field	Type	Description
id	String	A data value that identifies the event that caused the operation to be processed. This could be either a epc-tag or a trigger-urn if a trigger depending on what kind of matching event occurred.
opReports	List<PCOpReport>	An unordered list of PCOpReport instances, one for each of the corresponding PCOpSpec instances in the corresponding PCReportSpec in the corresponding order.
Stats	List<PCEventStat>	Null, if the statProfileNames parameter of the corresponding PCReportSpec is empty, omitted, or null. Otherwise, contains a PCEventStat for each statistics profile named in the statProfileNames parameter of the corresponding PCReportSpec, in the corresponding order.

Table 35: PCEventReport Fields

9.3.5 PCOpReport

A `PCOpReport` contains the result of a single `PCOpSpec` executing on a port during a port cycle.

PCOpReport
`state : Boolean // Conditional`
`opStatus : PCOpStatus`
`opName : String // Conditional`
`<<extension point>>`
`---`

This ALE implementation constructs a `PCOpReport` as follows:

Field	Type	Description
`state`	`Boolean`	(Conditional) The result of the operation, according to the table below, or null if an error occurred.
`opStatus`	`PCOpStatus`	Specifies whether the operation succeeded or failed (see Section 9.3.6).
`opName`	`String`	(Conditional) A copy of the `opName` parameter of the corresponding `PCOpSpec`. Omitted if the `opName` parameter was omitted from the corresponding `PCOpSpec`.

Table 36: PCOpReport Fields

The value of the `data` field is constructed according to the following table:

PCOpType Value	Description	state Value
`READ`	`Read port status.`	The boolean value that was read from the port status which indicates if the port is activated or not.
`WRITE`	`Set port status.`	The value is omitted.

Table 37: PCOpReport state Field Values

9.3.6 PCOpStatus

`PCOpStatus` is an enumerated value that lists the several possible outcomes for a given operation.

<<Enumerated Type>> PCOpStatus
SUCCESS
MISC_ERROR_TOTAL
MISC_ERROR_PARTIAL
PORT_NOT_FOUND_ERROR
OP_NOT_POSSIBLE_ERROR
<<extension point>>

The codes that contain ERROR in their names are errors that arise during the interaction between the ALE implementation and the Port.
This ALE implementation returns PCOpStatus codes according to the following table:

Status Code	Description
SUCCESS	The operation completed successfully.
MISC_ERROR_TOTAL	An error occurred during the processing of this operation that resulted in total failure. The state of the Port following the operation is unchanged. The ALE implementation returns this code only if no more specific code applies.
MISC_ERROR_PARTIAL	An error occurred during the processing of this operation that resulted in partial failure. The state of the Port following the operation attempt is indeterminate.
PORT_NOT_FOUND_ERROR	The specified port of the reader was not found.
OP_NOT_POSSIBLE_ERROR	The specified operation is not possible on the specified port or on the specified reader.

Table 38: PCOpStatus Values

9.3.7 PCEventStat

A PCEventStat provides additional, implementation-defined information about each ”occurring” of an event, which is, each time an event acquired by one of the Reader participating in the port cycle.

PCEventStat
profile : PCStatProfileName
statBlocks : List<ECReaderStat>
<<extension point>>

The ALE implementation constructs a PCEventStat as follows:

Field	Type	Description
profile	PCStatProfileName	The name of the statistics profile that governed the generation of this PCEventStat instance.
statBlocks	List<ECReaderStat>	An unordered list containing an ECReaderStat instance for each Reader that occurred this event.

Table 39: PCEventStat Fields

9.3.8 PCEventTimestampStat

PCEventTimestampStat is a subclass of PCEventStat. The ALE implementation includes one PCEventTimestampStat in a PCEventReport if the EventTimestamp statistics profile was included in the corresponding PCReportSpec.

PCEventTimestampStat includes all of the fields in PCEventStat, plus the following additional fields:

PCEventTimestampStat
firstOccurringTime : dateTime
lastOccurringTime : dateTime

The ALE implementation constructs a PCEventTimestampStat as follows:

Field	Type	Description
profile	PCStatProfileName	This field contains the EventTimestamp value of PCStatProfileName.
statBlock	List<ECReaderStat>	This field contains an empty list.
firstOccurringTime	dateTime	This field contains the first time within this port cycle that the event occurred by any reader contributing to this port cycle.
lastOccurringTime	dateTime	This field contains the last time within this port cycle that the event occurred by any reader contributing to this port cycle.

Table 40: PCEventTimestampStat Fields

9.3.9 PCEventCountStat

PCEventCountStat is a subclass of PCEventStat. The ALE implementation includes one PCEventCountStat in a PCEventReport if the EventCount statistics profile was included in the corresponding PCReportSpec. PCEventCountStat includes all of the fields in PCEventStat, plus the following additional fields:

PCEventCountStat
count : int

The ALE implementation constructs a PCEventCountStat as follows:

Field	Type	Description
profile	PCStatProfileName	This field contains the EventCount value of PCStatProfileName.
statBlock	List<ECReaderStat>	This field contains an empty list.
Count	int	This field contains the count how often this event occurred within this port cycle by any reader contributing to this port cycle.

Table 41: PCEventCountStat Fields

9.4 ALEPCCallback Interface

The ALEPCCallback interface is the path by which this ALE implementation delivers asynchronous results from port cycles to subscribers.

<<interface>> ALEPCCallback
callbackResults (reports : PCReports) : void

Referring to the state transition tables in Section 5.6.1 of the ALE specification Document, whenever a transition specifies that ”reports are delivered to subscribers” this ALE implementation attempts to deliver the results to each subscriber by invoking the callbackResults method of the ALEPCCallback interface once for each subscriber, passing the PCReports for the port cycle as specified by the notification URI for that subscriber as spec-

ified in the subscribe call. All subscribers receive an identical PCReports instance.

10 Bindings for the Callback APIs

This section specifies XML-based bindings for ALECallback, ALECCCallback and ALEPCCallback interfaces, through which the ALE Reading API, the ALE Writing API and the ccr Input and Output API, respectively, deliver asynchronous notifications to subscribers. Each binding of these interfaces specifies a syntax for notification URIs. A notification URI is supplied by an ALE client as a parameter of the subscribe and unsubscribe methods of the ALE Reading, Writing API or Digital Input and Output API. The notification URI both selects a binding of the callback interface to be used for that subscriber, and provides addressing information in a manner specified by each binding below.

10.1 HTTP Binding

The HTTP bindings of the ALECallback, ALECCCallback and ALEPCCallback interfaces provide for delivery of ECReports, CCReports or PCReports, respectively, in XML via the HTTP protocol using the POST operation.

The syntax for HTTP notification URIs as used by these bindings is defined in RFC2616. Informally, an HTTP URI has one of the two following forms:

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

where

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

The HARTING IT Software Development GmbH & Co KG vendor implementation delivers event cycle, command cycle or port cycle reports by sending an HTTP POST request to the callback receiver designated in the URI, where remainder-of-URL is included in the HTTP request-line, and where the payload is the ECReports instance, CCReports instance or PCReports instance encoded in XML according to the schema.

This ALE implementation does not interpret the response code value returned by the callback receiver. Therefore the implementation interprets any response code that is not null as a normal response, not indicative of any error.

10.2 TCP Binding

The TCP binding of the `ALECallback`, `ALECCCallback` and `ALEPCCallback` interfaces provide for delivery of `ECReports`, `CCReports` or `PCReports`, respectively, in XML via a raw TCP connection.

The syntax for TCP notification URIs as used by these bindings is defined in RFC2396. Informally, a TCP URI has the following form:

tcp://host:port

where

- `host` is the DNS name or IP address of the host where the callback receiver is listening for incoming TCP connections.
- `port` is the TCP port on which the callback receiver is listening for incoming TCP connections.

The HARTING IT Software Development GmbH & Co KG vendor implementation delivers event cycle, command cycle or port cycle reports by opening a new TCP connection to the specified host and port, writing to the connection the `ECREports` instance, `CCReports` instance or `PCReports` instance encoded in XML according to the schema and then closing the connection. The implementation does not require a reply or acknowledgement.

10.3 FILE Binding

The FILE bindingsof the `ALECallback`, `ALECCCallback` and `ALEPCCallback` interfaces provide for delivery of `ECReports`, `CCReports` or `PCReports`, respectively, in XML to a file.

The syntax for FILE notification URIs as used by these bindings is defined in RFC1738. Informally, a FILE URI has one of the two following forms:

`file:///path/file`

`file:///path/fileDATETIME`

where

- `path` is the pathname of a folder within the local filesystem.
- `file` is the filename of a file within the local filesystem.
- `DATETIME` is a placeholder within the `file` parameter which will be replaced by the current date time in format ”YYYYMMDDHHMMSSFF” this results in a new file for every delivered report. If omitted only one file with the name of the file parameter will be used.

The HARTING IT Software Development GmbH & Co KG implementation delivers an event cycle, command cycle or port cycle report by appending to the specified file within folder the `ECReports`, `CCReports` or `PCReports` instance encoded in XML according to the schema. The implementation will raise an `InvalidURIException` if the folder specified by the path parameter does not exists, if the file specified by the file parameter does not exists the implementation will create the file. If the DATETIME placeholder is used a new file will be created for every delivered `ECReports`, `CCReports` or `PCReports` instance instead of appending the report to a single file. Note

that if DATETIME is omitted and more than one cycle completes, the file will contain a concatenation of XML documents, rather than a single XML document.

10.4 SQL Binding

The HARTING IT Software Development GmbH & Co KG vendor implementation provides different SQL bindings of the `ALECallback`, `ALECCCallback` and `ALEPCCallback` interfaces for delivery of `ECReports`, `CCReports` or `PCReports`, in SQL statement via the an SQL INSERT operation. The following chapters will describe the different SQL bindings and in chapter 10.4.3 the mapping option for all these bindings will be described as well.

10.4.1 MySQL Binding

The syntax for MySQL notification URIs as used by these binding is defined below. Informally, a MySQL URI has one of two the following forms:

```
mysql://?connection=$ConnectionString$
        &table=$TableName$
        &plain=$ColumnName$

mysql://?connection=$ConnectionString$
        &table=$TableName$
        &$ColumnMappingList$
```

where

- `connection` specifies the connection string of the MySQL database (i.e `connection=HOST=localhost; DATABASE=db1;UID=scott;PWD=tiger`).
- `table` specifies a table name of the MySQL table within the database.
- `plain` specifies a column name within the table where the implementation will write to the `ECReports` instance, `CCReports` instance or the `PCReports` instance encoded in XML according to the schema. If omitted the column mapping will be used instead.
- `ColumnMappingList` is a list of query parameters that will map specific information from an `ECReports`, `CCReports` or `PCReports` instance into the specified column within the table. This ALE implementation will interpret these parameters as defined in section 10.4.3.

The HARTING IT Software Development GmbH & Co KG vendor implementation delivers event cycle, command cycle or port cycle reports by sending one or more MySQL INSERT operations to the specified table and columns within the database, and where the values are the `ECReports` instance, `CCReports` instance or `PCReports` instance information mapped on columns as specified trough the URI see section 10.4.3 for mapping options.

10.4.2 SQLITE Binding

The syntax for SQLite notification URIs as used by these binding is defined below. Informally, a SQLite URI has one of two the following forms:

```
sqlite://?connection=$ConnectionString$
        &table=$TableName$
        &plain=$ColumnName$

sqlite://?connection=$ConnectionString$
        &table=$TableName$
        &$ColumnMappingList$
```

where

- `connection` specifies the connection string of the MySQL database (i.e `Data Source=`c:/mydb.db;`Version=`3;`Password=`myPassword;).
- `table` specifies a table name of the SQLite table within the database.
- `plain` specifies a column name within the table where the implementation will write to the `ECReports` instance, `CCReports` instance or the `PCReports` instance encoded in XML according to the schema. If omitted the column mapping will be used instead.
- `ColumnMappingList` is a list of query parameters that will map specific information from an `ECReports`, `CCReports` or `PCReports` instance into the specified column within the table. This ALE implementation will interpret these parameters as defined in section 10.4.3.

The HARTING IT Software Development GmbH & Co KG vendor implementation delivers event cycle, command cycle or port cycle reports by sending one or more MySQL INSERT operations to the specified table and columns within the database, and where the values are the `ECReports` instance, `CCReports` instance or `PCReports` instance information mapped on columns as specified trough the URI see section 10.4.3 for mapping options.

10.4.3 SQL Binding Mapping Options

This ALE implementation provides the two mapping options plain and column mapping. If the plain parameter is specified the implementation will insert the complete `ECReports`, `CCReports` or `PCReports` instance encoded in XML according to schema in a single column within the table. If omitted the column mapping will be used instead, where one INSERT operation will be send to the database for each `ECReportGroupListMember` within an `ECReports`, `CCOpReport` within a `CCReports` or `PCOpReport` within a `PCReports`. The implementation recognizes the parameters defined in the next three sub-sections for the column mapping options.

10.4.3.1 ECReports

The HARTING IT Software Development GmbH & Co KG vendor implementation provides SQL binding mapping parameters for `ECReports` as specified

1399 in the below table.

Parameter	Related Information	Description
date	ECReports.date	If specified the date parameter within the ECReports will be inserted in the column according to the value. (i.e. date=DateColumn)
report	ECReports.ECReport. reportname	If specified the reportname parameter within the ECReport will be inserted in the column according to the value.(i.e. report=ReportNameColumn)
group	ECReports.ECReport. ECReportGroup.groupName	If specified the groupname parameter within the ECReporGroup will be inserted in the column according to the value.(i.e. group=GroupNameColumn)
count	ECReports.ECReport. ECReportGroup.groupCount	If specified the groupCount parameter within the ECReportGroup will be inserted in the column according to the value.(i.e. group=GroupNameColumn)
epc	ECReports.ECReport. ECReportGroup ECReportGroupListMember epc	If specified the epc parameter within the ECReportGroupListMember will be inserted in the column according to the value.(i.e. epc=EPCColumn)
tag	ECReports.ECReport. ECReportGroup ECReportGroupListMember tag	If specified the tag parameter within the ECReportGroupListMember will be inserted in the column according to the value.(i.e. tag=TagColumn)
rawHex	ECReports.ECReport. ECReportGroup ECReportGroupListMember rawHex	If specified the rawHex parameter within the ECReportGroupListMember will be inserted in the column according to the value.(i.e. rawHex=RawHexColumn)
field	ECReports.ECReport. ECReportGroup ECReportGroupListMember ECReportMemberField value	A parameter to specify a comma separated list of column names. If specified the value parameter of each member of the fieldList parameter within the ECReportGroupListMember will be inserted in order of their appearance in the next specified column. (i.e. field=Field1, Field2, Field3) If fiewer columns then fields are specified the remaining fields will be omitted from the INSERT operation.

1400 Table 42: ECReports MySQL Binding Parameter

Through the notification URI each parameter could be assigned to a column within the database table. If a parameter is omitted the related information from the `ECReports` will be omitted in the INSERT operation as well.

10.4.3.2 CCReports

The HARTING IT Software Development GmbH & Co KG vendor implementation provides SQL binding mapping parameters for `CCReports` as specified in the below table.

Parameter	Related Information	Description
`date`	`CCReports.date`	If specified the `date` parameter within the `CCReports` will be inserted in the column according to the value. (i.e. `date=DateColumn`)
`report`	`CCReports.CCCmdReport. cmdSpecName`	If specified the `cmdSpecName` parameter within the `CCCmdReport` will be inserted in the column according to the value.(i.e. `report=ReportNameColumn`)
`id`	`CCReports.CCCmdReport. CCTagReport.id`	If specified the `id` parameter within the `CCTagReport` will be inserted in the column according to the value.(i.e. `group=IdColumn`)
`name`	`CCReports.CCCmdReport. CCTagReport.CCOpReport opName`	If specified the `opName` parameter within the `CCOpReport` will be inserted in the column according to the value.(i.e. `name=NameColumn`)
`status`	`CCReports.CCCmdReport. CCTagReport.CCOpReport opStatus`	If specified the `opStatus` parameter within the `CCOpReport` will be inserted in the column according to the value.(i.e. `tag=StatusColumn`)
`data`	`CCReports.CCCmdReport. CCTagReport.CCOpReport data`	If specified the `data` parameter within the `CCOpReport` will be inserted in the column according to the value.(i.e. `data=DataColumn`)

Table 43: CCReports MySQL Binding Parameter

Through the notification URI each parameter could be assigned to a column within the database table. If a parameter is omitted the related information from the `CCReports` will be omitted in the INSERT operation as well.

10.4.3.3 PCReports

The HARTING IT Software Development GmbH & Co KG vendor implementation provides SQL binding mapping parameters for `PCReports` as specified in the below table.

Parameter	Related Information	Description
date	PCReports.date	If specified the date parameter within the PCReports will be inserted in the column according to the value. (i.e. date=DateColumn)
report	PCReports.PCReport. reportName	If specified the reportName parameter within the PCEventReport will be inserted in the column according to the value.(i.e. report=ReportNameColumn)
id	PCReports.PCReport. PCEventReport.id	If specified the id parameter within the PCEventReport will be inserted in the column according to the value.(i.e. group=IdColumn)
name	PCReports.PCReport. PCEventReport.PCOpReport opName	If specified the opName parameter within the PCOpReport will be inserted in the column according to the value.(i.e. name=NameColumn)
status	PCReports.PCReport. PCEventReport.PCOpReport opStatus	If specified the opStatus parameter within the PCOpReport will be inserted in the column according to the value.(i.e. tag=StatusColumn)
data	PCReports.PCReport. PCEventReport.PCOpReport data	If specified the data parameter within the PCOpReport will be inserted in the column according to the value.(i.e. data=DataColumn)

Table 44: PCReports MySQL Binding Parameter

Through the notification URI each parameter could be assigned to a column within the database table. If a parameter is omitted the related information from the PCReports will be omitted in the INSERT operation as well.

11 ALE Reader Configuration API

The ALE Reader Configuration API is an interface through which clients may obtain configuration information from base reader defined by the Logical Reader API. The API allows clients to obtain the complete configuration and not only the configuration properties, which were set through the Logical Reader API.

11.1 ALERC-Main API Class

<<interface>> ALERM
---
getRCConfiguration(name : String) : RCConfiguration
getStandardVersion() : String
getVendorVersion() : String
<<extension point>>

The HARTING IT Software Development GmbH & Co KG vendor implementation implements the methods of the ALE Reader Configuration API as specified in the following table:

Method	Argument /Result	Type	Description
getRCConfiguration	name	String	Returns the complete configuration of the base reader named name.
	[result]	RCConfiguration	
getStandardVersion	**[result]**	String	Returns a string that identifies what version of the specification this implementation of the Reader Configuration API complies with.
getVendorVersion	**[result]**	String	Returns a string that identifies what vendor extensions this implementation of the Reader Configuration API provides.

Table 45: ALERC Interface Methods

The primary data type associated with the ALE Reader Configuration API is the RCConfiguration, which represents the complete configuration of a given base reader. The next section, Section 11.2, specify the RCConfiguration data type in full detail.

11.1.1 Error Conditions

Methods of the ALE Reader Configuration API signal error conditions to the client by means of exceptions. The following exceptions are defined. All the exception types in the following table are extensions of a common ALEException base type, which contains one string element giving the rea-

1445 son for the exception.

Exception Name	Meaning
`SecurityException`	The operation was not permitted due to an access control violation or other security concern.
`NoSuchNameException`	The specified reader name does not exist.
`NonBaseReaderException`	The specified reader named name is not a known base reader.
`ImplementationException`	A generic exception raised by the implementation for reasons that are implementation-specific. This exception contains one additional element: a severity member whose values are either ERROR or SEVERE. ERROR indicates that the ALE implementation is left in the same state it had before the operation was attempted. SEVERE indicates that the ALE implementation is left in an indeterminate state.

1446 Table 46: Exceptions in the ALERC Interface

1447 The exceptions that may be raised by each ALERC method are indicated in
1448 the table below. This ALE implementation raises the appropriate exception
1449 listed below when the corresponding condition described above occurs. If
1450 more than one exception condition applies to a given method call, the Ale
1451 implementation may raise any of the exceptions that applies.

ALE Method	Exceptions
`getRCConfiguration`	NoSuchNameException NonBaseReaderException SecurityException ImplementationException
`getStandardVersion`	ImplementationException
`getVendorVersion`	ImplementationException

1452 Table 47: Exceptions Raised by each ALEPC Interface Method

1453 11.2 RCConfiguration

1454 An RCConfiguration describes a complete configuration of a single base
1455 reader.

RCConfiguration
name : String
xmlStyleSheet: RCStyleSheet
node:List<RCNode>
<<extension point>>

The HARTING IT Software Development GmbH & Co KG vendor implementation constructs an RCConfiguration as follows:

Field	Type	Description
name	String	The name of the base reader that was requested using the getRCConfiguration method.
xmlStyleSheet	RCStyleSheet	An object with style information to display the configuration.
node	List<RCNode>	An unordered list of configuration nodes, each representing a configuration branch or leaf.

Table 48: RCConfiguration Fields

11.3 RCStyleSheet

An RCStyleSheet contains information to display an RCConfiguration instance according to the reader type of a base reader.

RCStyleSheet
type : String
href : String
<<extension point>>

The HARTING IT Software Development GmbH & Co KG vendor implementation constructs an RCStyleSheet as follows:

Field	Type	Description
type	String	Indicates what kind of style sheet document is linked in the href parameter.
href	String	A link to a style sheet document to display an RCConfiguration according to the reader type.

Table 49: RCStyleSheet Fields

11.4 RCNode

A RCNode represents a configuration node of a base reader. An RCNode is identified by a name and contains other RCNodes and/or RCProperties, it is structured such like an XML document.

RCNode
name : String
properties : List<RCProperty>
node : List<RCNode>
<<extension point>>

The HARTING IT Software Development GmbH & Co KG vendor implementation constructs an RCNode as follows:

Field	Type	Description
name	String	Represents the name of this configuration node according to the reader type.
properties	List<RCProperty>	An unordered list of reader configuration properties corresponding to this node.
node	List<RCNode>	An unordered list of child nodes corresponding to this node..

Table 50: RCNode Fields

11.5 RCProperty

An RCProperty represents a configuration property of a base reader.

RCProperty
name : String
value : String
<<extension point>>

The HARTING IT Software Development GmbH & Co KG vendor implementation constructs an RCProperty as follows:

Field	Type	Description
name	String	The name of the reader configuration property.
value	String	The value of the reader configuration property.

Table 51: RCProperty Fields

12 Appendix

13 Glossary

14 References

Literary References

–

Internet References

- The ALE Specification Part 1, EPCglobal
  http://www.gs1.org/gsmp/kc/epcglobal/ale/ale_1_1_1-standard-core-20090313.pdf
  EPCglobal, Version 1.1.1
- The ALE Specification Part 2, EPCglobal
  http://www.gs1.org/gsmp/kc/epcglobal/ale/ale_1_1_1-standard-XMLandSOAPbindings-20090313.pdf
  EPCglobal, Version 1.1.1

