Conformance Requirements Document

Version 1.0.4

for Application Level Events (ALE) 1.1.1

This version was approved by the SAG Filtering and Collection WG and TSC on January 10, 2010

Disclaimer

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

Abstract

This document outlines the approach to conformance testing for the EPCglobal Application Level Events (ALE) 1.1 specification. The objective of an ALE conformance certification program is to test and certify solution providers' implementations of the EPCglobal ALE interface v1.1. Certification of ALE conformance provides confidence for buyers in the operational capability of a specific product's implementation of the ALE interfaces, while providing solution providers a benchmark to assure product functionality.

Status of this document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. The latest status of this document series is maintained at the EPCglobal. This document has been reviewed and approved per the steps outlined in the Standards Development Process for errata. It is to be used in conjunction with the ratified ALE 1.1 specification as published on the EPCglobal website.
This latest version of 1.0.4 is based on correcting errata discovered duing beta testing and certification efforts at MET Labs and approved by the Filtering and Collection Working Group on December 17, 2009 and subsequently by the TSC on January 10, 2010.

For a complete list of all the latest comments received that were reviewed and evaluated to update this version of the ALE 1.1 Conformance Requirements, consult the following file available only to EPCglobal Subscribers: *ALE_1_1-Conformance-Req-Issues_20081218.pdf* which will be made available from the ALE standards page from the following URL: http://www.epcglobalinc.org/standards/ale.

Some of changes made to the previous version or version 1.0 of the Conformance Requirements for ALE 1.1 are as follows:

- TCR-R7 Step 3: Correction → initiationCondition changed to initiationTrigger
- TCR-R9 Step 20: Correction → "expiry of N sections" changed to "immediately"
- TCR-R9 Step 21: Clarification note added
- TCR-R17 Pre-test Conditions: ECFieldSpec7 removed since is was not used
- TCR-R18 Step 5: Correction → Changed ECReportOutputFieldSpecC to ECReportOutputFieldSpecD
- TCR-W3: Deleted Step 31 since it repeated a previous step. Renumbered steps starting at 30. Previously there was no step 30.
- TCR-W4 Pre-test Conditions: filterList value changed to "no" in CCSpec
- TCR-W5 Pre-test Conditions: filterList value changed to "no" in all CCSpecs
- TCR-W6 Pre-test Conditions: filterList value changed to "no" in CCSpec
- TCR-W7 Pre-test Conditions: filterList value changed to "no" in CCSpec
- TCR-W8 Pre-test Conditions: filterList value changed to "no" in CCSpec and
- TCR-W8 Step 2: deleted "and RawHex" from expected results.

- TCR-W9 Pre-test Conditions: filterList value changed to “no” in CCSpec
- TCR-W10 Pre-test Conditions: filterList value changed to “no” in CCSpecs
- TCR-W11 Pre-test Conditions: First bullet corrected and filterList value changed to “no” in CCSpec
- TCR-W12 Pre-test Conditions: First bullet corrected and filterList value changed to “no” in CCSpec
- TCR-W13 Pre-test Conditions: filterList value changed to “no” in CCSpec
- TCR-W14 Pre-test Conditions: filterList value changed to “no” in CCSpec
- TCR-W14 Pre-test Conditions: CCOpSpec List B FieldSpec changed from @1.96 to @1.96.32
- TCR-T3 Step 3: clarification note added
- TCR-T4: Inserted new step after step 2 to correct oversight of not defining the ECSpec. Subsequent steps were renumbered.
- TCR-T5: Inserted new step after step 2 to correct oversight of not defining the ECSpec. Subsequent steps were renumbered.
- TCR-A3 Pre-test Conditions: Updated ACPermission5
- TCR-A3 Step 32: Added clarification note
- TCR-A3 Step 34: Correction:Changed expected result to ACPermission3
- TCR-A3 Step 40: Correction:Changed expected result to ACPermission4
- TCR-A3 Step 44: Correction: Changed ECSpec to LRSpec
- TCR-A3 Step 46: ACPermission1 to ACPPermission5
- TCR-A3 Step 50: Correction: Changed ECSpec to LRSpec
- TCR-A3 Step 52: ACPermission1 to ACPPermission6
- TCR-A3 Step 53: ACClientIdentity3 to ACClientIdentity2
- TCR-A3 Step 54: No exceptions should be raised.
- TCR-A3 Step 54: Rewrote step to add clarity
- TCR-A4 Step 7: Added clarification note
- TCR-A4 Step 19: changed “unknown” to “invalid”
- TCR-A4 Step 20: Added clarification note
- TCR-A4 Step 28: Added clarification note
- TCR-L2 Step 2: clarified expected results
- TCR-L2 Step 4: changed isComposite to true since API defined readers are not being tested.
- TCR-L2 Step 5: clarified expected results
- TCR-L2 Step 11: clarified expected results
- TCR-L4 Step 6: clarified expected results
- TCR-L5 Pre-test Conditions: clarified bullet
- TCR-L5 Step 2: changed isComposite to true since API defined readers are not being tested.
- TCR-L5 Step 5: Changed LRSpec name to LR1
- TCR-L5 Step 19: Clarified that a valid logical reader name should be used
- TCR-L5 Steps 20, 21, 23, 24 and 31: Clarified the how the expected results are different when an API-defined readers are supported.

CONTENTS

1 Introduction

Technical implementations of the Application Level Events (ALE) specification may vary due to distinct interpretations of the specification and/or use of proprietary technologies when developing systems that implement the EPCglobal Architecture Framework. Conformance testing provides a mechanism to ensure that solutions adhere to, and are compatible with, the specified standard. An Application Level Events (ALE) Conformance Certification Program provides solution providers a benchmark to assure product functionality according to the ALE specification, while imparting confidence on potential buyers in the operational capability of a specific product's implementation of the ALE interface.

ALE certification represents an endorsement that helps solution provider differentiate their products and services within the marketplace. Certification of ALE conformance instills both product recognition and a level of public confidence sought by corporate supply chains looking to partner with a solution provider of EPCglobal standard compliant products. Implementation of an ALE certification program will:

- Help move the industry toward RFID Interoperability
- Accelerate ALE and EPC Implementations
- Publicly identify product vendors who support the EPCglobal standards.

The focus of this program will be both software and hardware product conformance to the EPCglobal ALE 1.1 Interface Specification. The Application Level Events (ALE) specification describes an interface through which client applications may obtain filtered, consolidated EPC data from a variety of sources or perform operations on data carriers (e.g. RFID Tags) such as writing data, reading data or killing tags.

The design of the interface recognizes that in most EPC processing systems, there is a level of processing that reduces the volume of data that comes directly from the EPC data source, such as an RFID reader, into "coarser" events of interest to applications. An ALE interface provides summations of EPC data that higher level business applications can consume and interpret. Also, the interface provides a level of abstraction to ease the burden on EPC processing systems when performing operations on data cariers. In particular, the ALE 1.1 specification is designed to provide full access to the functionality of the EPCglobal UHF Class 1 Gen 2 [Gen2] specification, when interacting with Gen2 RFID Tags.

The EPCglobal Filtering and Collection working group is responsible for defining the ALE Certification test scenarios that the authorized testing agency will use in developing a test harness and associated test scripts.

2 Scope

An ALE Conformance Certification Program will focus on testing a given application's implementation of the ALE Interface and its conformance to the ALE 1.1 Specification.

Test case requirements and benchmark definitions will be developed by the EPCglobal Filtering and Collection working group.

An ALE Conformance Certification Program is NOT intended to test the performance, reliability, or scalability of the tested product.

3 Program Overview

The ALE Certification Program will be offered by a certified testing laboratory to solution providers enrolled in the certification program.

Program Implementation and Certificate definition are to be defined by EPCglobal US and a chosen Testing Laboratory.

An EPCglobal ALE Conformance Certification Program will focus on testing the following aspects of the ALE interface:

- Support all EPCglobal tag encoding as defined in the EPCglobal Tag Data Standard, version 1.3.1
- Support of the ALE event cycle model as it pertains to reader cycles and the inclusion of tags in EC Reports.
- Support of the ALE command cycle model as it pertains to reader cycles and the inclusion of tags in CC Reports.
- Support of all methods defined in the ALE APIs.
- Support all exceptions conditions defined in the ALE APIs.
- Support of the XML and SOAP bindings for the ALE APIs.
- Support of the HTTP, HTTPS, FILE and TCP notification mechanisms for ALE Subscriptions; when provided by an ALE implementation.
- Support for each parameter of the ECSpec and those permutations identified as important to validate conformance.
- Support for each parameter of the CCSpec and those permutations identified as important to validate conformance.
- Support for each parameter of the TMSpec and those permutations identified as important to validate conformance.
- Support for Fieldnames, Datatypes, and Formats
- Validation of ECReports for conformance with format specification.
- Validation of CCReports for conformance with format specification.
- Correct implementation of the extensibility mechanisms defined for the XML and SOAP bindings of the ALE APIs.

The conformance tests may not be exhaustive, but should be representative of capabilities needed for a successful ALE implementation. The tests should be defined to be platform independent, and should not require products to be implemented on any particular system or platform.

The EPCglobal Filtering and Collection group has envisioned 3 classes of product that may implement the ALE specification and should be included in the conformance test design: Software, Hardware, and Other Products.

Product Class	Product Description
Software	A software product (e.g. RFID Middleware) that processes tag reads and commands from an arbitrary number of external reader devices
Hardware	A Hardware Product (e.g. Smart RFID reader) that embeds an ALE implementation.
Other	Specialized implementations of ALE that are not capable of reading arbitrary RFID Tags (e.g. a barcode scanner that implements ALE)

4 Terminology

This document adopts terminology developed by the World Wide Web Consortium [W3C-Conformance]:

- *Certificate Issuer* The organization that issues certificates of conformance, namely, EPCglobal.
- *Testing Laboratory* An organization that carries out certification testing on behalf of the Certificate Issuer
- *Specification* An EPCglobal specification for which conformance is tested.
- *Implementation Under Test (IUT)* A submission of hardware and/or software for which certification is sought by an EPCglobal subscriber.
- *System Under Test (SUT)* The IUT together with any other apparatus required to carry out the test.
- *Test Method* A description of the test that is applied to the SUT. There may be more than one Test Method available for a given ALE 1.1 specification requirement, each providing a different level of conformance testing.
- *Test Report* Quoting from [W3C-Conformance]: "A Test Report contains the results of the testing effort. The test report should provide enough information that, if necessary, the testing effort could be duplicated. The testing report should contain:
  - a complete description of the IUT,
  - the name of the Testing Laboratory,
  - the signature of a Testing Laboratory official,
  - the date that the testing was completed,

- the name and version number of the Test Method
- the results of the Test Method
- an unambiguous statement indicating pass or fail."

- *ALE Conformance Certification Program* An EPCglobal US sponsored Software/Hardware solution certification program measuring ALE 1.1 conformance.
- *Certificate of Conformance* Quoting again from [W3C-Conformance]: "The certificate of conformance is typically a summation of the Test Report. Since it is often used in the procurement process, it includes information most pertinent between the buyer and the seller."

5 Submission Requirements

Solution providers who wish to submit their product(s) for testing must submit the following to the testing laboratory:

- An Implementation Under Test (IUT). This may take one of the following forms:
  - Software that implements both the server ALE interface 1.1 and the Reader & Trigger Simulator interfaces. The software tests may be applied over the internet to the implementation under test, so that physical installation is not necessary to complete the test cases.
  - Software that implements both the ALE interface and the Reader Simulator & Trigger Simulator interfaces. Where software is compatible with platforms supported by the testing laboratory, the submission may be in the form of a CD-ROM plus written installation instructions. Otherwise, the software must be submitted pre-installed on compatible hardware, with written instructions for starting and shutting down the hardware.
  - Any other kind of system that implements the ALE interface, including (but not limited to) ALE implementations embedded in RFID readers or other devices.

6 ALE 1.1 General Functional Requirements

The ALE 1.1 defines specific functionality that a valid ALE implementation must provide regardless of which APIs are implemented The following tables outline the specific requirements that must be tested as defined by the ALE 1.1 specification. Each test requirement entry references the ALE 1.1 Specification and the test case requirement (TCR) used to verify functionality as defined in Sections 14 through 18 of this document.

6.1 General API Mandatory Requirements Matrix

The following table outlines the mandatory general requirements for an ALE implementation as defined by the ALE 1.1 Specification. All mandatory general requirements have a requirement number of GMx where x is a decimal number. The Protocol Sub-Clause is from Part 1 of the ALE 1.1 specification unless otherwise noted. Any requirement whose requirement number has an asterisk (*) following it is optional and only tested if an implementation has implemented the feature. Only Gen2 tags will

be used for conformance testing. Requirements for Gen1 tags will have "N/A" in the "How Verified" column.

Req. No.	Protocol Sub-Clause	Requirements (Requirements, Command, …)	Applies to (ref)	How Verified (by Demonstration or by Design)
GM1	4	To comply with this specification, an implementation of a given ALE API SHALL fully implement that API according to this specification.		By Demonstration: TCR: All test cases for each API implemented.
GM2	4.3	An ALE implementation SHALL implement these methods as specified in the Table 2: getStandardVersion, getVendorVersion. The result returned by each method SHALL only pertain to the API to which it belongs.		By Demonstration: TCR – R1, W1, T1, L1, A1
GM3	4.3	A getStandardVersion implementation SHALL return a string corresponding to a version of this specification to which the API implementation fully complies. Version 1.1 of the ALE specification, the implementation SHALL return the string `1.1`		By Demonstration: TCR – R1, W1, T1, L1, A1
GM4	4.3	A getVendorVersion returning an empty string SHALL indicate that the implementation implements only standard functionality of the API with no vendor extensions. The result returned by this method SHALL only pertain to the API to which it belongs.		By Demonstration: TCR - R1, W1, T1, L1, A1
GM5	4.3	When an implementation chooses to return a non-empty string for getVendorVersion, the value returned SHALL be a URI where the vendor is the owning authority. The result returned by this method SHALL only pertain to the API to which it belongs.		By Demonstration: TCR - R1, W1, T1, L1, A1 to confirm a valid URI. By Design to confirm owning Authority: TCR-R21, W16, T6, A5, L7
GM6	4.5	Except as noted elsewhere in this specification, an ALE implementation SHALL accept as a name any non-empty string of Unicode characters that does not include Pattern_White_Space or		By Demonstration TCR – R2, R3, W2, W3

		Pattern_Syntax characters (as those classes are defined in [Unicode])		
GM7	4.5	An ALE implementation SHALL consider the specified name equivalent to the previously specified name if it is an identical sequence of Unicode characters.		By Demonstration TCR – R2. W2
GM8	4.5	An ALE implementation SHALL NOT consider the specified name equivalent to the previously specified name if they are not canonical equivalent sequences (except in situations of aliasing explicitly noted elsewhere in this specification).		By Demonstration TCR – R3, W3
GM9	4.6	An ALE implementation SHALL permit the same string to be used as a name in more than one namespace. (see table in section 4.6)		By Demonstration: TCR – R5, W4
GM10	4.7	Within this specification, the terms "null," "omitted," and "empty string" are used interchangeably to denote an absent value. An implementation SHALL NOT draw any distinction between "null," "omitted," and "empty string." If a binding provides more than one representation as illustrated above, the ALE implementation SHALL treat them as equivalent.		By Demonstration TCR – Various Reading and Writing API test cases. The test device should use variations of null, omitted and empty string throughout the test cases.
GM11	4.7	An implementation SHALL NOT draw any distinction between an omitted list and a list containing zero elements. If a binding provides more than one representation for this situation, the ALE implementation SHALL treat them as equivalent.		By Demonstration TCR - Various Reading and Writing API test cases. The test device should use variations of null, omitted and empty string throughout the test cases.
GM12	5.6.1	An EC/CCSpec that is created by a call to the `define` method of the ALE Reading/Writing API SHALL begin in the *unrequested* state,		By Demonstration TCR – R2, W2

		with an empty set of subscribers.		
GM13	5.6.1	An EC/CCSpec that is created by a call to the `define` method SHALL be subject to the state transitions specified in the three tables 6, 7 and 8		By Demonstration TCR – R4, R5, W4, W5, W6
GM14	5.6.1	The transitions in table 6 SHALL apply when the EC/CCSpec is in the *unrequested* state.		By Demonstration TCR - R4, R5, W4, W5, W6
GM15	5.6.1	The transitions in table 7 SHALL apply when the EC/CCSpec is in the *requested* state.		By Demonstration TCR - R4, R5, W4, W5, W6
GM16	5.6.1	The transitions in Table 8 SHALL apply when the EC/CCSpec is in the *active* state.		By Demonstration TCR - R4, R5, W4, W5, W6
GM17	5.6.1	A call to undefined from the active state reports SHALL have `terminationCondition` set to `UNDEFINE`. For an ECSpec, the reports SHALL include any Tags that were read prior to the `undefine` call. For a CCSpec, the reports SHALL include any operations that were completed prior to the `undefine` call.		By Demonstration TCR – R19, W15
GM18	5.6.1	Events occuring at times other than those specified in the tables 6, 7 and 8 SHALL NOT cause a state transition		By Demonstration TCR - R4, R5, W4, W5, W6
GM19	5.6.1	If an ALE Writing API implementation receives a second `poll` call for a CCSpec for which there is already an outstanding `poll` call, and the second `poll` call specifies different parameter values, the ALE implementation SHALL satisfy the second `poll` by a initiating a new command cycle rather than sharing the results of the first, as though the second `poll` were of a different CCSpec.		By Demonstration TCR – W7
GM20	5.6.1	Simultaneous `poll` calls for the same CCSpec that specify no parameters SHALL share the same command cycle, as implied by the state diagrams in this section.		By Demonstration TCR – W7
GM21	5.6.2	An EC/CCSpec that is created by a call to the `immediate` method of the ALE		By Demonstration

		Reading/Writing API SHALL begin in the *requested* state if any start triggers are specified, and in the *active* state if no start triggers are specified		TCR – R6, W8.
GM22	5.6.2	An EC/CCSpec that is created by a call to the `immediate` method SHALL be subject to the state transitions specified in the Tables 9 and 10.		By Demonstration TCR – R6, W8
GM23	5.6.2	The transitions a given in Table 9 SHALL apply when the EC/CCSpec is in the *requested* state.		By Demonstration TCR – R6, W8
GM24	5.6.2	The transitions given in Table 10 SHALL apply when the EC/CCSpec is in the *active* state.		By Demonstration TCR – R6, W8
GM25	5.6.2	Events occuring at times other than those specified in the Tables 9 and 10 SHALL NOT cause a state transition.		By Demonstration TCR – R6,W8
GM26	6.1	An ALE implementation SHALL recognize each fieldname defined in this section and interpret it as defined herein (see Section 6.1).		By Demonstration TCR – R8, R17, W14
GM27*	6.1	An ALE implementation that implements the TMSpec API SHALL recognize fieldnames defined through that API (see Section 7).		By Demonstration TCR – R17, W14
GM28	6.1.1	An ALE implementation SHALL recognize the string `epc` as a valid fieldname as specified in this section		By Demonstration TCR – R8, R17, W5
GM29	6.1.1	When interacting with a Gen2 Tag, an ALE implementation SHALL interpret the `epc` fieldname as referring to the EPC/UII content of the EPC memory bank (Bank 01_2) as defined in [Gen2].		By Demonstration TCR – R8, R17, W14
GM30*	6.1.1	When interacting with a Gen1 Tag, an ALE implementation SHALL interpret the `epc` fieldname as referring to the EPC content of the Tag; that is, the EPC payload (the number of bits being fixed by the tag) not including CRC or other non-EPC bits. The treatment SHALL be equivalent to a Gen2 tag whose toggle bit (bit 17h) and Reserved/AFI bits (bits 18h-1Fh) are zeros.		N/A
GM31	6.1.1	When interacting with a Gen1 or Gen2 Tag, an ALE implementation SHALL raise an "operation not possible" condition if an		By Demonstration TCR – W5

		attempt is made to carry out a "lock" operation on the `epc` field.		
GM32	6.1.1	If a fieldspec specifies a fieldname of `epc` and specifies any other datatype besides `epc`, the ALE implementation SHALL consider the fieldspec to be invalid.		By Demonstration TCR—R3, W3
GM33	6.1.2	An ALE implementation SHALL recognize the string `killPwd` as a valid fieldname as specified in this section.		By Demonstraion TCR – R8, R17. W14
GM34	6.1.2	When interacting with a Gen2 Tag, an ALE implementation SHALL interpret the `killPwd` fieldname as a synonym for the fieldname `@0.32`, that is, for offset 00_h to $1F_h$ in the RESERVED memory bank of a Gen2 Tag, which holds the Kill Password.		By Demonstration TCR – R8, R17, W14
GM35	6.1.2	The default datatype for the `killPwd` field SHALL be `uint`; the default format SHALL be `hex`. The implementation SHALL NOT permit any other datatypes defined in this specification to be used for the `killPwd` field.		By Demonstration TCR- R8, R17, W14
GM36	6.1.3	An ALE implementation SHALL recognize the string `accessPwd` as a valid fieldname as specified in this section.		By Demonstation TCR – R8, R17. W14
GM37	6.1.3	When interacting with a Gen2 Tag, an ALE implementation SHALL interpret the `accessPwd` fieldname as a synonym for the fieldname `@0.32.32`, that is, for offset 20_h to $3F_h$ in the RESERVED memory bank of a Gen2 Tag, which holds the Access Password.		By Demonstration TCR – R8, R17, W14
GM38	6.1.3	The default datatype for the `accessPwd` field SHALL be `uint`; the default format SHALL be `hex`. The implementation SHALL NOT permit any other datatypes defined in this specification to be used for the `accessPwd` field.		By Demonstration TCR – R8, R17, W14, R3, W3
GM39	6.1.4	An ALE implementation SHALL recognize the string `epcBank` as a valid fieldname as specified in this section		By Demonstraton TCR – R8, R17, W14
GM40	6.1.4	When interacting with a Gen2 Tag, an ALE implementation SHALL interpret the `epcBank` fieldname as referring to the		By Demonstration TCR – R8, R17,

		content of the EPC memory bank (Bank 01_2) as defined in [Gen2].		W14
GM41	6.1.4	If the implementation cannot or does not wish to support reading to the end of the memory bank, an ALE implementation SHALL raise an "operation not possible" condition when an attempt is made to read from the `epcBank` field.		By Demonstration TCR – R8
GM42	6.1.4	The default datatype for the `epcBank` field SHALL be `bits` ; the default format SHALL be `hex`. The implementation SHALL NOT permit any other datatypes defined in this specification to be used for the `epcBank` field.		By Demonstration TCR – R8, R17, W14, R3, W3
GM43	6.1.5	An ALE implementation SHALL recognize the string `tidBank` as a valid fieldname as specified in this section.		By Demonstration TCR – R8, R17, W14
GM44	6.1.5	When interacting with a Gen2 Tag, an ALE implementation SHALL interpret the `tidBank` fieldname as referring to the content of the TID memory bank (Bank 10_2) as defined in [Gen2].		By Demonstration TCR – R8, R17, W14
GM45	6.1.5	If the implementation cannot or does not wish to support reading to the end of the memory bank, an ALE implementation SHALL raise an "operation not possible" condition when an attempt is made to read from the `tidBank` field.		By Demonstration TCR – R8
GM46	6.1.5	The default datatype for the `tidBank` field SHALL be `bits`; the default format SHALL be `hex`. The implementation SHALL NOT permit any other datatypes defined in this specification to be used for the `tidBank` field.		By Demonstration TCR – R8, R17, W14, R3, W3
GM47	6.1.6	An ALE implementation SHALL recognize the string `userBank` as a valid fieldname as specified in this section.		By Demonstration TCR – R8, R17, W14
GM48	6.1.6	When interacting with a Gen2 Tag, an ALE implementation SHALL interpret the `tidBank` fieldname as referring to the content of the TID memory bank (Bank 10_2) as defined in [Gen2].		By Demonstration TCR – R8, R17, W14
GM49	6.1.6	If the implementation cannot or does not wish		By

		to support reading to the end of the memory bank, an ALE implementation SHALL raise an "operation not possible" condition when an attempt is made to read from the `userBank` field.		Demonstration TCR – R8
GM50	6.1.6	The default datatype for the `userBank` field SHALL be `bits`; the default format SHALL be `hex`. The implementation SHALL NOT permit any other datatypes defined in this specification to be used for the `userBank` field.		By Demonstration TCR – R8, R17, W14, R3, W3
GM51	6.1.7	An ALE implementation SHALL recognize the string `afi` as a valid fieldname as specified in this section.		By Demonstration TCR – R8, W14
GM52	6.1.7	When interacting with a Gen2 Tag, an ALE implementation SHALL interpret the `afi` fieldname as a synonym for the fieldname `@1.8.24`, that is, for offset 18_h to $1F_h$ in the EPC/UII memory bank of a Gen2 Tag, which may hold the ISO 15962 Application Family Identifier (AFI).		By Demonstration TCR – R8, W14
GM53*	6.1.7	When interacting with a Gen1 Tag, an ALE implementation SHALL interpret the `afi` fieldname as a "field not found".		N/A
GM54	6.1.7	When interacting with a Gen2 Tag, an ALE implementation SHALL raise an "operation not possible" condition if an attempt is made to carry out a "lock" operation on the `afi` field.		By Demonstration TCR – R8, W14
GM55	6.1.7	The default datatype for the `afi` field SHALL be `uint`; the default format SHALL be `hex`. The implementation SHALL NOT permit any other datatypes defined in this specification to be used for the `afi` field.		By Demonstration TCR – R8, W14, R3, W3
GM56	6.1.8	An ALE implementation SHALL recognize the string `nsi` as a valid fieldname as specified in this section		By Demonstration TCR – R8, W14
GM57	6.1.8	When interacting with a Gen2 Tag, an ALE implementation SHALL interpret the `nsi` fieldname as a synonym for the fieldname `@1.9.23`, that is, for offset 17_h to $1F_h$ in the EPC/UII memory bank of a Gen2 Tag, which holds the Numbering System Identifier (NSI).		By Demonstration TCR – R8, W14
GM58*	6.1.8	When interacting with a Gen1 Tag, an ALE implementation SHALL interpret the `nsi`		N/A

		fieldname as a “field not found”.		
GM59	6.1.8	When interacting with a Gen2 Tag, an ALE implementation SHALL raise an “operation not possible” condition if an attempt is made to carry out a “lock” operation on the `nsi` field.		By Demonstration TCR – R8, W14
GM60	6.1.8	The default datatype for the `nsi` field SHALL be `uint`; the default format SHALL be `hex`. The implementation SHALL NOT permit any other datatypes defined in this specification to be used for the `nsi` field.		By Demonstration TCR – R8, W14, R3, W3
GM61	6.1.9	An ALE implementation SHALL recognize any string beginning with an @ character as a valid fieldname as specified by the syntax in the following sub-sections, provided that the string also meets the constraints: The *`bank`* portion must be `0` or a positive integer with no leading zeros. The *`length`* portion must be a positive integer with no leading zeros. The *`offset`* portion (if specified) must be `0` or a positive integer with no leading zeros.		By Demonstration TCR – R17, W14
GM62	6.1.9	An ALE implementation SHALL consider any string beginning with an `@` character but not conforming to any syntax specified herein, or not meeting the constraints stated in GM61 as an invalid fieldname.		By Demonstration TCR – R3
GM63	6.1.9.1	An ALE implementation SHALL recognize any string of the form `@`*`bank`*`.`*`length`*[`.`*`offset`*] as a valid fieldname as specified in this section, provided that the string also meets the constraints as stated in GM61.		By Demonstration TCR – R17, W14
GM64	6.1.9.1	An ALE implementation SHALL consider any string beginning with an `@` character but not conforming to this syntax, or not meeting the as stated in GM61, as an invalid fieldname.		By Demonstration TCR – R3, W3
GM65	6.1.9.1	An ALE implementation SHALL interpret an absolute address fieldname as a fixed field comprising *`length`* contiguous bits starting at offset *`offset`* within memory bank *`bank`*.		By Demonstration TCR – R17, W14
GM66	6.1.9.1	If *`offset`* is omitted, the ALE implementation SHALL treat the fieldname in the same way as if *`offset`* were `0`		By Demonstration TCR – R17, W14

GM67	6.1.9.1	When interacting with a Gen2 Tag, an ALE implementation SHALL interpret ***bank*** as given in Table 11		By Demonstration TCR – R8, R17, W14
GM68	6.1.9.1	Any other ***bank*** value not in Table 11 SHALL result in a "field not found" condition when interacting with a Gen2 Tag.		By Demonstration TCR – R17, W14
GM69	6.1.9.1	When interacting with a Gen2 Tag, the fieldname SHALL be interpreted as referring to the contiguous field whose most significant bit is ***offset*** and whose least significant bit is bit (***offset*** + ***length*** – 1), following the addressing convention specified in [Gen2].		By Demonstration TCR – R8, R17, W14
GM70*	6.1.9.1	When interacting with a Gen1 Tag, an ALE implementation SHALL interpret a ***bank*** of `0` as referring to the EPC memory of the Tag. Any other ***bank*** value SHALL result in a "field not found" condition when interacting with a Gen1 Tag. The ***offset*** field SHALL be interpreted as referring to an offset from the most significant bit of tag memory, and the fieldname SHALL be interpreted as referring to the contiguous field whose most significant bit is ***offset*** and whose least significant bit is bit (***offset*** + ***length*** – 1), following that addressing convention.		N/A
GM71	6.1.9.1	The default datatype for absolute address fieldnames is `uint`. The default format for absolute address fieldnames is `hex`. The set of legal datatypes for an absolute address fieldname SHALL be the set of datatypes for which binary encoding and decoding is defined, that is, `uint`, `bits`, `epc` and any implementation-specific datatypes that support binary encoding and decoding.		By Demonstration TCR – R8, R17, W14
GM72	6.1.9.2	An ALE implementation SHALL recognize any string of the form `@`***bank.oid*** as a valid fieldname as specified in this sub-section, provided that the string also meets the constraints: The ***bank*** portion must be `0` or a positive integer with no leading zeros. The ***oid*** portion must be a valid Object Identifier represented in the URN syntax specified in [RFC3061].		By Demonstration TCR – R18, W14

GM73	6.1.9.2	An ALE implementation SHALL interpret a variable fieldname as a variable field, referring to an ISO 15962 "data set" whose Object Identifier is ***oid*** and which is encoded in Tag memory using the encoding rules specified in [ISO15962].		By Demonstration TCR – R18, W14
GM74	6.1.9.2	When interacting with a Gen2 Tag, an ALE implementation SHALL interpret ***bank*** as given in Table 12.		By Demonstration TCR - R18, W14
GM75	6.1.9.2	Any ***bank*** value, other than those in Table 12, SHALL result in a "field not found" condition when interacting with a Gen2 Tag		By Demonstration TCR – R18, W14
GM76*	6.1.9.2	When interacting with a Gen1 Tag, an ALE implementation SHALL result in a "field not found" condition when referring to an ISO data set.		N/A
GM77	6.1.9.2	An implementation MAY choose not to support variable fieldnames for WRITE operations, in which case an attempt to do so SHALL raise an "operation not possible" condition.		By Demonstration TCR – W14
GM78	6.1.9.2	An implementation MAY also choose not to support variable fieldnames for READ operations and for the Reading API, in which case an attempt to do so SHALL raise an "operation not possible" condition.		By Demonstration TCR – R18
GM79	6.1.9.3	An ALE implementation SHALL recognize variable pattern fieldnames as specified in this section. A variable pattern fieldname has the form `@`***`bank`***`.`***`oid-prefix`***`.*`, where ***bank*** is as specified in section 6.1.9.2 , and ***oid-prefix*** is a string conforming to the URN syntax for OIDs specified in [RFC3061].		By Demonstartion TCR – R18, W14
GM80*	6.1.9.3	When an `ECReportOutputFieldSpec` (section 8.2.11) includes a variable pattern fieldname, the ALE implementation SHALL report all ISO 15962 data sets in the specified memory bank whose OID has ***oid-prefix*** as a prefix.		By Demonstration TCR – R18, W14
GM81*	6.1.9.3	The fieldname appearing in the `ECReportMemberField` (section 8.3.7) instance corresponding to each data set SHALL be a variable fieldname (section 6.2.9.2) containing the full OID of the data set		By Demonstration TCR – R18, W14

		(unless overridden by a non-null `name` parameter in the `ECReportOutputFieldSpec`).		
GM82	6.2	An ALE implementation SHALL recognize each datatype and format defined in section 6.2 and interpret it as defined herein		By Demonstration TCR – R8, R17, W14
GM83	6.2	An ALE implementation SHALL consider a fieldspec invalid if the format is not compatible with the datatype, or if the format is a read-only format and the fieldspec is being used in a context that requires a read-write format		By Demonstartion TCR – R3, W3
GM84	6.2.1	An ALE implementation SHALL recognize the string `epc` as a valid datatype as specified in section 6.2.1		By Demonstration TCR – R8, R17, W4, W14
GM85	6.2.1	The encoding and decoding of the `epc` datatype SHALL be according to the EPCglobal Tag Data Standard [TDS1.3.1].		By Demonstration TCR – R8, R17, W5, W14
GM86	6.2.1.1	When reading and writing values of the `epc` datatype in a field that includes a toggle bit and AFI (including the `epc` field as specified in section 6.1.1), decoding and encoding SHALL take place as specified in section 6.2.1.2 .		By Demonstration TCR – R8, R17, W5, W14
GM87	6.2.1.1	When reading and writing values of the `epc` datatype in a field that does not include a toggle bit and AFI (including an absolute address field as specified in section 6.1.9.1), the following rule applies. Decoding SHALL take place as specified in section 6.2.1.2 , using the rules for the case where the toggle bit and the AFI are not available.		By Demonstation TCR – R8, R17, W5, W14
GM88	6.2.1.1	Encoding SHALL take using those same rules from Section 6.2.1.2 where the toggle bit and the AFI are not available, with the following modifications: • If the encoded value has more bits than are available in the specified field, an "out of range" condition SHALL be raised. • If the encoded value has fewer bits than are available in the specified field, the encoded value SHALL be padded with		By Demonstration TCR – R8, R17, W5, W14

		trailing zero bits to fit. That is, the most significant bit of the encoded value is aligned to the most significant bit of the field, and the least significant bits of the field beyond the encoded value are filled with zeros. If the EPC value is of the form `urn:epc:raw:N.A.V`, an "out of range" condition SHALL be raised (because there is no available toggle and AFI, required for values of this form).		
GM89	6.2.1.2	An ALE implementation SHALL recognize the format names specified in Table 13 and permit their use with the `epc` datatype. The notation "RW" below indicates that the ALE implementation SHALL permit the format in both reading and writing contexts, while the notation "RO" indicates that the ALE implementation SHALL permit the format only in reading contexts		By Demonstration TCR – R8, R17, W5, W14
GM90	6.2.1.3	An ALE implementation SHALL recognize pattern syntax as specified in Table 14 for each of the formats defined for use with the `epc` datatype		By Demonstration TCR – R8, R17, W5, W14
GM91	6.2.1.4	An ALE implementation SHALL recognize grouping pattern syntax as specified in Table 15 for each of the formats defined for use with the `epc` datatype.		By Demonstration TCR – R8, R17, W5, W14
GM92	6.2.1.4	EPC grouping patterns SHALL be interpreted as given in Table 16, 17, 18 and 19.		By Demonstation TCR – R16
GM93	6.2.2.	An ALE implementation SHALL recognize the string `uint` as a valid datatype as specified in section 6.2.2		By Demonstration TCR – R8, R17, W5, W14
GM94	6.2.2.1	When converting between a sequence of N bits and a value of type `uint`, the leftmost bit SHALL be considered to be the most significant bit		By Demostration TCR – R8, R17, W5, W14
GM95	6.2.2.1	If an `uint` value to be encoded to a sequence of N bits is greater than or equal to 2^N, an "out of range" condition SHALL be raised		By Demonstration TCR – W14
GM96	6.2.2.2	An ALE implementation SHALL recognize `hex` and `decimal` as valid formats for the `uint` datatype, as specified in section 6.2.2.2		By Demonstration TCR – R8, R17,

				W5, W14
GM97	6.2.2.2	For output, the ALE implementation SHALL construct a `HexUnsignedInteger` string with no leading zeros, except that the value zero itself is represented by a single '`0`' digit. The string SHALL NOT contain lowercase letters		By Demonstration TCR – R8, R17, W5, W14
GM98	6.2.2.2	For input, the ALE implementation SHALL accept any `HexUnsignedInteger` string		By Demonstration TCR – R8, R17, W5, W14
GM99	6.2.2.2	For output, the ALE implementation SHALL construct a `DecimalUnsignedInteger` string with no leading zeros, except that the value zero itself is represented by a single '`0`' digit		By Demonstration TCR – R8, R17, W5, W14
GM100	6.2.2.2	For input, the ALE implementation SHALL accept any `DecimalUnsignedInteger` string		By Demostration TCR – R8, R17, W5, W14
GM101	6.2.2.3	An ALE implementation SHALL recognize pattern syntax as specified in section 6.2.2.3 for each of the formats defined for use with the `uint` datatype		By Demonstration TCR – R8, R17, W5, W14
GM102	6.2.2.3	An ALE implementation SHALL interpret these patterns as follows for both formats. If a pattern is a single integer value (i.e., `HexUnsignedInteger` or `DecimalUnsignedInteger` as appropriate), the pattern matches a value equal to the pattern. If a pattern is the '`*`' character, the pattern matches any value. If a pattern is in the form `[`*`lo`*`-`*`hi`*`]`, the pattern matches any value between *`lo`* and *`hi`*, inclusive. If a pattern is in the form `&`*`mask`*`=`*`compare`* the pattern matches any value that is equal to *`compare`* after being bitwise and-ed with *`mask`*		By Demonstration TCR – R8, R17, W5, W14
GM103	6.2.2.4	An ALE implementation SHALL recognize grouping pattern syntax as specified in section 6.2.2.4 for each of the formats defined for use with the `uint` datatype		By Demonstration TCR – R8, R17, W5, W14
GM104	6.2.2.4	Unsigned grouping patterns SHALL be interpreted as given in Tables 20 and 21 plus explanatory text		By Demonstration. TCR – R8, R17,

				W5, W14
GM105	6.2.3	An ALE implementation SHALL recognize the string `bits` as a valid datatype as specified in section 6.2.3		By Demonstration TCR – R8, R17, W5, W14
GM106	6.2.3.1	When reading a value of type `bits`, the ALE implementation SHALL return the unmodified sequence of bits read from the field		By Demonstration TCR – R8, R17, W5, W14
GM107	6.2.3.1	When writing a value of type bits, table 22 SHALL be used based on the number of bits in the of the `bits` value (M) and the number of bits in the field (N):		By Demonstration TCR – R8, R17, W5, W14
GM108	6.2.3.2	An ALE implementation SHALL recognize `hex` as a valid format for the `bits` datatype		By Demonstration TCR – R8, R17, W5, W14
GM109	6.2.3.2	For output, the ALE implementation SHALL construct the length part without leading zeros. The bit pattern SHALL be represented using N `HexDigit` characters, where N is the length divided by 4 and rounded up to the next higher integer, padding with leading zero bits as necessary. The string SHALL NOT contain lowercase letters		By Demonstration TCR – R8, R17, W5, W14
GM110	6.2.3.2	For input, the ALE implementation SHALL accept any `HexBits` string where the length specified in the first part of the `HexBits` string, divided by 4 and rounded up to the next higher integer, matches the number of `HexDigit` characters in the second part. If the length is not divisible by 4, the ALE implementation SHALL require the input to be padded with leading zero bits		By Demonstration TCR – R8, R17, W5, W14
GM111	6.2.4	An ALE implementation SHALL recognize the string `iso-15962-string` as a valid datatype referring to a string of zero or more characters drawn from the Unicode character set [Unicode], encoded according to ISO 15962 [ISO15962].		By Demonstration TCR – R8, R17, W5, W14
GM112	6.2.4.1	An ALE implementation SHALL recognize `string` as a valid format for the `iso-15962-string` datatype. In the `string` format, a string is represented simply as a sequence of Unicode characters		By Demonstration TCR – R8, R17, W5, W14

		corresponding directly to the characters encoded in the Tag		

7 ALE 1.1 Reading API Functional Requirements

The ALE 1.1 defines specific functionality that a valid ALE Reading API implementation must provide. The following tables outline the specific requirements that must be tested as defined by the ALE 1.1 specification. Each test requirement entry references the ALE 1.1 Specification and the test case requirement (TCR) used to verify functionality as defined in section 14 of this document.

7.1 Reading API Mandatory Requirements Matrix

The following table outlines the mandatory requirements for an ALE Reading implementation as defined by the ALE 1.1 Specification. All mandatory Reading API requirements have a requirement number of RMx where x is a decimal number. Any requirement whose requirement number has an asterisk (*) following it is optional and only tested if an implementation has implemented the feature.

Req. No.	Protocol Sub-Clause	Requirements (Requirements, Command, …)	Applies to (ref)	How Verified (by Demonstration or by Design)
RM1	8.1	An ALE implementation SHALL implement the methods of the ALE Reading API as specified Table 28.		By Demonstration TCR – R1 through R19
RM2	8.1	getECSpec Returns the ECSpec that was provided when the ECSpec named specName was created by the define method. The result SHALL be equivalent to the ECSpec that was provided to the define method, but NEED NOT be identical.		By Demonstration TCR – R2
RM3*	8.1.1	If the Reading API implementation is associated with an implementation of the Access Control API (Section 11), the Reading API implementation SHALL raise the SecurityException if the client was not granted access rights to the called method as specified in Section 11.		By Demonstration TCR – A3
RM4	8.1.1	An ALE implementation SHALL raise the appropriate exception listed in Table 30 when the corresponding condition described in Table 29 occurs.		By Demonstration TCR – R3
RM5	8.2	The ALE implementation SHALL interpret the fields of an ECSpec given in Table 31		By Demonstration TCR – R2, R3, R5, R6

RM6	8.2	If includeSpecInReports is true, it specifies that each ECReports instance generated from this ECSpec SHALL include a copy of the ECSpec. If false, each ECReports instance SHALL NOT include a copy of the ECSpec.		By Demonstration TCR – R6, R7, R13
RM7		The define and immediate methods SHALL raise an ECSpecValidationException if any of the following are true for an ECSpec instance: • The logicalReaders parameter is null, omitted, is an empty list, or 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 an ECSpecValidationException as specified in Section 8.2.1. • The reportSpecs parameter is null, omitted, empty, 2115 or any of the members of reportSpecs leads to an ECSpecValidationException as specified in Section 8.2.5. • Any member of the specified primaryKeyFields is not a known fieldname. • The implementation does not support the specified primaryKeyFields value with the specified logical readers.		By Demonstration TCR – R3
RM8	8.2	As an ALE implementation accumulates Tags during an event cycle, the implementation SHALL consider two Tags to be the same if both tags have the exact same values in all of the primary key fields. The ALE implementation SHALL also use the same rule to determine equality in implementing the ADDITIONS and DELETIONS values of ECReportSetSpec (Section 8.2.6) and the reportOnlyOnChange feature of ECReportSpec (Section 8.2.5).		By Demostration TCR – R10
RM9	8.2	If accessing any of the primary key fields on a Tag causes a “field not found” or “operation not possible” condition, then		By Demonstration TCR – R17

		that Tag SHALL be omitted from the event cycle.		
RM10	8.2	If the primaryKeyFields parameter is empty or omitted, the ALE implementation SHALL behave as though primaryKeyFields was set to a list containing the single element epc		By Demonstration TCR – R5, R6, R7, R9
RM11	8.2.1	The ALE implementation SHALL interpret the fields of an ECBoundarySpec as given in Table 32.		By Demonstration TCR – R5, R6, R7, R9, R11, R12, R14, R15
RM12	8.2.1	The define and immediate methods SHALL raise an ECSpecValidationException if any of the following are true for an ECBoundarySpec instance: • A negative number is specified for any of the ECTime 2173 values duration, repeatPeriod, and stableSetInterval. • The value of the startTrigger or stopTrigger, or 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.*, stopTrigger is omitted or null, stopTriggerList is empty, duration is zero or omitted, stableSetInterval is zero or omitted, whenDataAvailable is false, and no vendor extension stopping condition is specified.		By Demonstration TCR – R3
RM13	8.2.2	The ALE implementation SHALL interpret the fields of an ECTime instance as given in Table 33		By Demonstration TCR – R5
RM14	8.2.3	The ALE implementation SHALL interpret an instance of ECTimeUnit as specified in Table 34.		By Demonstration TCR – R5
RM15	8.2.4	An implementation SHALL raise an ECSpecValidationException if presented with a URI beginning with urn:epcglobal: that is not valid according		By Demonstration TCR – R3

		to this specification or any other EPCglobal specification that defines a standardized trigger URI.		
RM16	8.2.4	If a URI not beginning with `urn:epcglobal:` is not valid according to the implementation-specific rules, the implementation SHALL raise an ECSpecValidationException.		By Demonstration TCR – R3
RM17*	8.2.4.1	If an implemtation supports the Real-time Clock Standardized Trigger, the ALE implementation SHALL conform to the following specification for all such URIs valid according to the specification below and Table 35. A real-time clock trigger takes one of the two following forms: • urn:epcglobal:ale:trigger:rtc:*period.offset* • urn:epcglobal:ale:trigger:rtc:*period.offset.timezone* where *period*, *offset*, and *timezone* are as specified in Table 35		By Demonstration TCR – R20
RM18*	8.2.4.1	If an ALE implementation chooses to implement the Real-time Clock Standardized Trigger, it SHALL interpret a trigger of this form as follows. The trigger is delivered each time the number of milliseconds past midnight modulo *period* equals *offset*. "Midnight" refers to midnight in the specified time zone, which if omitted defaults to some implementation dependent default value		By Demonstration TCR – R20
RM19	8.2.5	The ALE implementation SHALL interpret the fields of an ECReportSpec as given in Table 36		By Demonstration TCR – R5, R6, R7, R8, R9, R10, R11
RM20	8.2.5	The define and immediate methods SHALL raise an ECSpecValidationException if any of the following are true for an ECReportSpec instance: • The specified reportName is an empty string or is not accepted by the implementation according to Section 4.5.		By Demonstration TCR – R3

		• The specified reportName is a duplicate of another report name in the same ECSpec. • The specified filterSpec leads to an ECSpecValidationException as specified in Section 8.2.7. • The specified groupSpec leads to an ECSpecValidationException as specified in Section 8.2.9. • The specified output leads to an ECSpecValidationException as specified in Section 8.2.10. • Any element of statProfileNames is not the name of a known statistics profile.		
RM21	8.2.5	An ECReports instance SHALL include an ECReport instance corresponding to each ECReportSpec in the governing ECSpec, in the same order specified in the ECSpec, except that an ECReport instance SHALL be omitted under the following circumstances: • If an ECReportSpec has reportIfEmpty set to false, then the corresponding ECReport instance SHALL be omitted from the ECReports for this event cycle if the final, filtered set of Tags is empty (i.e., if the final Tag list would be empty, or if the final count would be zero). • If an ECReportSpec has reportOnlyOnChange set to true, then the corresponding ECReport instance SHALL be omitted from the ECReports for this event cycle if the filtered set of Tags is identical to the filtered prior set of Tags, where equality is tested by considering the primaryKeyFields as specified in the ECSpec (see Section 8.2), and where the phrase 'the prior set of Tags' is as defined in Section 8.2.6. This comparison takes place before the filtered set has been modified based on reportSet or output parameters.		By Demonstration TCR - R5, R6, R7, R8, R9, R10, R11

		The comparison also disregards whether the previous ECReports was actually sent due to the effect of this parameter, or the reportIfEmpty parameter.		
RM22	8.2.5	When the processing of reportIfEmpty and reportOnlyOnChange results in *all* ECReport instances being omitted from an ECReports for an event cycle, then the delivery of results to subscribers SHALL be suppressed altogether. That is, a result consisting of an ECReports having zero contained ECReport instances SHALL NOT be sent to a subscriber.		By Demonstration TCR – R15
RM23	8.2.5	An ECReports instance SHALL always be returned to the caller of immediate or poll at the end of an event cycle, even if that ECReports instance contains zero ECReport instances.		By Demonstration TCR – R5, R6, R7, R8, R13, R18
RM24	8.2.5	The statProfileNames parameter is a list of ECStatProfileName, each of which corresponds to a statistics profile that will be included in the ECReports. If the ALE engine does not recognize any name in the list it SHALL raise an ECSpecValidationException.		By Demonstration TCR – R3
RM25	8.2.6	An ALE implementation SHALL interpret an instance of ECReportSetSpec as specified in Table 37.		By Demonstration TCR - R5, R6, R7, R8, R9, R10, R11
RM26	8.2.6	The meaning of "the prior set of Tags" is as follows. For a given subscriber to an ECSpec, beginning with the second event cycle to be completed after the subscribe call, the prior set of Tags SHALL refer to the set of Tags read during the immediately previous event cycle for that ECSpec. For the first event cycle to be completed after the subscribe call for a given subscriber, and for a poll call, the prior set of Tags SHALL refer to either the set of Tags read during some previous event cycle for that ECSpec, or the empty set, at the discretion of the implementation.		By Demonstration TCR – R7, R9, R10, R12 , R15

RM27	8.2.7	The ALE implementation SHALL interpret the fields of an ECFilterSpec as given in Table 38		By Demonstration TCR – R16, R17
RM28	8.2.7	The define and immediate methods SHALL raise an ECSpecValidationException if any of the following are true for an ECFilterSpec instance: • Any element of includePatterns is not a syntactically valid epc-tag pattern as specified in Section 6.2.1.3. • Any element of excludePatterns is not a syntactically valid epc-tag pattern as specified in Section 6.2.1.3. • Any element of filterList leads to an ECSpecValidationException as specified in Section 8.2.8.		By Demonstration TCR – R3
RM29	8.2.7	A Tag SHALL be included in the final report if it passes the test specified by *every* ECFilterListMember in filterList, as defined below *[sic]* (i.e the last part of Section 8.2.7)		By Demonstration TCR – R16, R17
RM30	8.2.8	The ALE implementation SHALL interpret the fields of an ECFilterListMember as given in Table 39		By Demonstration TCR – R16, R17
RM31	8.2.8	The define and immediate methods SHALL raise an ECSpecValidationException or CCSpecValidationException (in the Reading API or the Writing API, respectively) if any of the following are true for any ECFilterListMember instance: • The specified fieldspec is invalid (see Section 8.2.12). • The patList is empty. • Any element of patList does not conform to the syntax rules for patterns implied by the specified fieldspec.		By Demonstration TCR – R3, W3
RM32	8.2.9	The ALE implementation SHALL interpret the fields of an ECGroupSpec as given in Table 40.		By Demonstration TCR – R16
RM33	8.2.9	The define and immediate methods		By Demonstration

		SHALL raise an ECSpecValidationException if any of the following are true for an ECGroupSpec instance: • The specified fieldspec is invalid (see Section 8.2.12). • The specified fieldspec implies a datatype and format for which no grouping pattern syntax is defined. • Any element of patternList does not conform to the syntax rules for grouping patterns implied by the specified fieldspec. • The elements of patternList are not disjoint, according to the definition of disjointedness defined by the datatype and format implied by the specified fieldspec.		TCR – R3
RM34	8.2.9	Every filtered Tag that is part of an event cycle SHALL be assigned to exactly one group for purposes of reporting.		By Demonstration TCR – R16, R17
RM35	8.2.9	If the field value matches one of the grouping patterns in patternList, the group name SHALL be computed from the field value according to the formula specified in the definition of the datatype and format implied by fieldspec.		By Demonstration TCR-
RM36	8.2.9	If the field value does not match any of the grouping patterns in patternList, or if accessing the field causes a “field not found” or “operatio not possible” condition, the Tag SHALL be assigned to a special “default group.”		By Demonstration TCR-
RM37	8.2.9	The name of the default group SHALL be null.		By Demonstration TCR – R5
RM38	8.2.9	If the pattern list is empty (or if the group parameter of the ECReportSpec is null or omitted), then all Tags SHALL be assigned to the default group.		By Demonstration TCR-
RM39	8.2.10	If any of includeEPC, includeTag, includeRawHex, or includeRawDecimal are true, or if fieldList is non-empty, the ALE implementation SHALL set the groupList parameter of each ECReportGroup instance to an ECReportGroupList instance, which in		By Demostration TCR – R6, R7, R8 R15, R16

		turn SHALL contain a list of ECReportGroupListMember instances having parameters set according to the table below. Otherwise, the ALE implementation SHALL set the groupList parameter to null.		
RM40	8.2.10	If includeCount is true, the ALE implementation SHALL set the groupCount parameter of each ECReportGroup instance to an ECReportGroupCount instance, with parameters set according to the table below. Otherwise, the ALE implementation SHALL set the groupCount parameter to null.		By Demonstration TCR – R7, R8, R13, R16, R17
RM41	8.2.10	The ALE implementation SHALL interpret the fields of an ECReportOutputSpec as given in Table 41.		By Demonstration TCR - R6, R7, R8 R15, R16
RM42	8.2.10	The define and immediate methods SHALL raise an ECSpecValidationException if any of the following are true for any ECReportOutputSpec instance: • Two members of fieldList have the same name (after applying defaults as specified in Section 8.2.11). • Any member of fieldList has a fieldspec parameter that is an invalid ECFieldSpec (see Section 8.2.12). • All five booleans includeEPC, includeTag, includeRawHex, includeRawDecimal, and includeCount are false, fieldList is empty or omitted, and there is no vendor extension to ECReportOutputSpec.		By Demonstration TCR – R3
RM43	8.2.11	The ALE implementation SHALL interpret the fields of an ECReportOutputFieldSpec as given in Table 41		By Demonstration TCR - R6, R7, R8 R15, R16
RM44	8.2.12	An ALE implementation SHALL interpret an ECFieldSpec instance as given in Table 42.		By Demonstration TCR – R18
RM45	8.2.12	An ALE implementation SHALL consider an ECFieldSpec instance		By Demonstration TCR – R17

		invalid if any of the following are true: • The value of fieldname is not a valid absolute address fieldname as defined in Section 6.1.9.1, a valid variable fieldname as defined in Section 6.1.9.2, a valid variable pattern fieldname as defined in Section 6.1.9.3, the name of a built-in fieldname as defined in Section 6.1 or otherwise provided by the ALE implementation as a vendor extension, or a user-defined fieldname defined via the Tag Memory API (Section 7). • The value of fieldname is a valid variable pattern fieldname as defined in Section 6.1.9.3, but the ECFieldSpec instance is in some context other than an ECReportOutputFieldSpec instance. • The value of datatype is not a valid datatype for the specified fieldname. • The value of format is not a valid format for the specified fieldname and specified datatype (or the default datatype for the specified fieldname, if datatype is omitted).		
RM46	8.2.14	The define and immediate methods of the ALE API (Section 8.1) SHALL raise an ECSpecValidationException if any of the conditions in the bulleted list in Section 8.2.14 are true.		By Demonstation TCR – R3
RM47	8.3	The ECReports implementation SHALL include these fields according to Table 44		By Demonstration TCR – R5, R6, R7
RM48	8.3.1	The ALE implementation SHALL set the initiationCondition field of an ECReports instance generated at the conclusion of an event according to the condition that caused the event cycle to start, as specified in Table 45.		By Demonstration TCR - R5, R6, R7, R19
RM49	8.3.2	The ALE implementation SHALL set the terminationCondition field of an ECReports instance generated at the conclusion of an event cycle according to the condition that caused the event		By Demonstration TCR - R5, R6, R7

		cycle to end, as specified Table 46.		
RM50	8.3.3	An ALE implementation SHALL construct an ECReport as given in Table 47.		By Demonstration TCR - R5, R6, R7
RM51	8.3.4	An ALE implementation SHALL construct an ECReportGroup as given in Table 48.		By Demonstration TCR - R5, R6, R7
RM52	8.3.5	An ALE implementation SHALL construct an ECReportGroupList as given in Table 49.		By Demonstration TCR - R5, R6, R7
RM53	8.3.5	Each distinct Tag included in this group SHALL have a distinct ECReportGroupListMember element in the ECReportGroupList, even if those ECReportGroupListMember elements would be identical due to the fields and formats selected.		By Demonstration TCR - R5, R6, R7
RM54	8.3.5	If both tags are read in the same event cycle, and ECReportOutputSpec specified includeEPC true and all other formats false, then the resulting ECReportGroupList SHALL have two ECReportGroupListMember elements, each having the same pure identity URI in the epc field.		By Demonstration TCR – R16
RM55	8.3.5	Similarly, if two Tags have the same values in one or more user defined fields, and ECReportOutputSpec only specified reading from those fields, the resulting ECReportGroupList SHALL have two ECReportGroupListMember elements, each having the same user fields in the fieldList parameter.		By Demonstration TCR – R16
RM56	8.3.6	An ALE implementation SHALL construct an ECReportGroupListMember from information read from a single Tag, as given in Table 50		By Demonstration TCR - R5, R6, R7
RM57	8.3.7	An ALE implementation SHALL construct an ECReportMemberField as given in Table 51		By Demonstration TCR - R5, R6, R7
RM58	8.3.8	An ALE implementation SHALL construct an ECReportGroupCount as given in Table 52.		By Demonstration TCR - R5, R6, R7
RM59	8.3.9	An ALE implementation SHALL construct an ECTagStat as given in		By Demonstration TCR – R6

		Table 53		
RM60	8.3.10	An ALE implementation SHALL construct an ECReaderStat as given in Table 54		By Demonstration TCR – R6
RM61	8.3.12	An ALE implementation SHALL include one ECTagTimestampStat in an ECReportGroupListMember if the TagTimestamps statistics profile was included in the corresponding ECReportSpec and the implementation chooses to implement the TagTimestamps statistics profile.		By Demonstration TCR - R6
RM62	8.3.12	An ALE implementation SHALL construct an ECTagTimestampStat as given in Table 55		By Demonstration TCR – R6
RM63*	8.3.12	Implementations MAY choose to use any clock that they wish to measure firstSightingTime and lastSightingTime, but they SHALL correct for any differences in clocks such that those time stamps are brought into synchronization with the date field of ECReports.		By Demonstration TCR – R6
RM64	8.4	Referring to the state transition tables in Section 5.6.1, whenever a transition specifies that "reports are delivered to subscribers" the ALE implementation SHALL attempt to deliver the results to each subscriber by invoking the callbackResults method of the ALECallback interface once for each subscriber, passing the ECReports for the event cycle as specified above, and using the binding and addressing information specified by the notification URI for that subscriber as specified in the subscribe call.		By Demonstration TCR – R4, R7, R9, R10, R11, R12, R14, R15

8 ALE 1.1 Writing API Functional Requirements

The ALE 1.1 defines specific functionality that a valid ALE Writng API implementation must provide. The following tables outline the specific requirements that must be tested as defined by the ALE 1.1 specification. Each test requirement entry references the ALE 1.1 Specification and the test case requirement (TCR) used to verify functionality as defined in section 15 of this document.

8.1 Writing API Mandatory Requirements Matrix

The following table outlines the mandatory requirements for an ALE Writng implementation as defined by the ALE 1.1 Specification. All mandatory Writing API requirements have a requirement number of WMx where x is a decimal number. Any requirement whose requirement number has an asterisk (*) following it is optional and only tested if an implementation has implemented the feature. Only Gen2 tags will be used for conformance testing. Requirements only for Gen1 tags will have "N/A" in the "How Verified" column.

Req. No.	Protocol Sub-Clause	Requirements (Requirements, Command, …)	Applies to (ref)	How Verified (by Demonstration or by Design)
WM1	9.1	An ALE implementation SHALL implement the above methods of the ALE Writing API as specified in Table 56.		By Demonstration TCR – W1 though W15
WM2	9.1	The lifecycle of a new CCSpec SHALL be subject to the provisions of Section 5.6.1		By Demonstration TCR – W2, W4, W5, W6, W7, W8
WM3	9.1	The CCSpec returned in the getCCSpec result SHALL be equivalent to the CCSpec that was provided to the define method, but NEED NOT be identical.		By Demonstration TCR – W2
WM4	9.1	If the Writing API implementation is associated with an implementation of the Access Control API (Section 11), the Writing API implementation SHALL raise this exception if the client was not granted access rights to the called method as specified in Section 11.		By Demonstration TCR – A3
WM5	9.1	An ALE implementation SHALL raise the appropriate exception listed in Table 58 when the corresponding condition described in Table 57 occurs.		By Demonstration TCR – W3
WM6	9.3	The ALE implementation SHALL interpret the fields of a CCSpec as given in Table 59.		By Demonstration TCR – W4, W5, W6, W7, W8, W9, W10, W11, W12, W13, W14
WM7	9.3	If includeSpecInReports is true, specifies that each CCReports instance generated from this CCSpec SHALL include a copy of the CCSpec		By Demonstration TCR – W8
WM8	9.3	The define and immediate methods SHALL raise a CCSpecValidationException if any of the following are true for a CCSpec instance:		By Demonstration TCR – W3

		• The logicalReaders parameter is null, omitted, is an empty list, or 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 CCSpecValidationException as specified in Section 9.3.1. • The cmdSpecs parameter is null, omitted, empty, or any of the members of cmdSpecs leads to a CCSpecValidationException as specified in Section 9.3.2.		
WM9	9.3.1	The ALE implementation SHALL interpret the fields of a CCBoundarySpec as given in Table 60.		By Demonstration TCR - W4, W5, W6, W7, W8, W9, W10, W11, W12, W13, W14
WM10	9.3.1	The define and immediate methods SHALL raise a CCSpecValidationException if any of the following are true for a CCBoundarySpec instance: • A negative number is specified for any of the ECTime values duration, repeatPeriod, or noNewTagsInterval. • 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]. • A negative number is specified for tagsProcessedCount. • No stopping condition apart from afterError is specified; *i.e.*, stopTriggerList is empty, duration is zero or omitted, noNewTagsInterval is zero or omitted, tagsProcsssedCount is zero or omitted, and no vendor extension stopping condition is specified.		By Demonstration TCR – W3
WM11	9.3.2	The ALE implementation SHALL interpret the fields of an CCCmdSpec as		By Demonstration TCR - W4, W5,

		given Table 61		W6, W7, W8, W9, W10, W11, W12, W13, W14
WM12	9.3.2	The ALE implementation SHALL process each Tag that matches filterSpec acquired during a command cycle in a manner equivalent to carrying out the operations specified in opSpecs in the order specified.		By Demonstration TCR – W14
WM13	9.3.2	The define and immediate methods SHALL raise a CCSpecValidationException if any of the following are true for a CCCmdSpec instance: • The specified name is an empty string or is not accepted by the implementation according to Section 4.5. • The specified name is a duplicate of another CCCmdSpec name in the same CCSpec. The specified filterSpec leads to a CCSpecValidationException as specified in Section 9.3.3. • The specified opSpecs leads to a CCSpecValidationException as specified in Section 9.3.4. • Any element of statProfileNames is not the name of a known statistics profile.		By Demonstration TCR – W3
WM14	9.3.2	A CCReports instance SHALL include an CCReport instance corresponding to each CCCmdSpec in the governing CCSpec, in the same order specified in the CCSpec, except that a CCReport instance SHALL be omitted under the following circumstance: • If a CCReportSpec has reportIfEmpty set to false, then the corresponding CCReport instance SHALL be omitted from the CCReports for this command cycle if the final, filtered set of Tags is empty (i.e., if there are no Tags to operate upon).		By Demonstration TCR – W5, W6, W7, W8, W9, W10, W11, W12, W13, W14, W15
WM15	9.3.2	When the processing of reportIfEmpty results in *all* CCReport instances being omitted from a CCReports for a command		By Demonstration TCR – W4, W5, W10

		cycle, then the delivery of results to subscribers SHALL be suppressed altogether. That is, a result consisting of a CCReports having zero contained CCReport instances SHALL NOT be sent to a subscriber.		
WM16	9.3.2	A CCReports instance SHALL always be returned to the caller of immediate or poll at the end of a command cycle, even if that CCReports instance contains zero CCReport instances.		By Demonstration TCR – W6, W7, W8, W9
WM17	9.3.3	The ALE implementation SHALL interpret the fields of a CCFilterSpec as given in Table 62.		By Demonstration TCR – W5, W6, W7, W8, W9, W10, W11, W12, W13, W14, W15
WM18	9.3.3	The define and immediate methods SHALL raise a CCSpecValidationException if any of the following are true for a CCFilterSpec instance: • Any element of filterList is leads to a CCSpecValidationException as specified in Section 8.2.8.		By Demonstration TCR – W3
WM19	9.3.3	A Tag SHALL be subject to the operations specified in the CCCmdSpec if it passes the test specified by *every* ECFilterListMember in filterList, as defined in Sections 8.2.7 and 8.2.8.		By Demonstration TCR – W5, W6, W7, W8, W9, W10, W11, W12, W13, W14, W15
WM20	9.3.3	If accessing a field specified by any element of filterList causes a "field not found" or "operation not possible" condition, that Tag SHALL not be processed as part of this CCCmdSpec.		By Demonstration TCR – W14
WM21	9.3.4	The ALE implementation SHALL interpret the fields of a CCOpSpec as given in Table 63.		By Demonstration TCR – W5, W6, W7, W8, W9, W10, W11, W12, W13, W14, W15
WM22	9.3.4	The define and immediate methods SHALL raise a CCSpecValidationException if any of the following are true for a CCOpSpec instance: • The specified opType value is not one of the standard opType values		By Demonstration TCR – W3

		specified in Section 9.3.5, or an implementation-specific value known to the ALE implementation. • The specified opType requires a fieldspec, and fieldspec is null or omitted. • The specified opType does not require a fieldspec, and fieldspec is specified. • The specified fieldspec is invalid according to Section 8.2.12. • The specified opType requires a dataSpec, and dataSpec is null or omitted. • The specified opType does not require a dataSpec, and dataSpec is specified. • The specified dataSpec is invalid according to Section 9.3.6. • The specified dataSpec specifies a value that is invalid for the specified operation, as specified in Section 9.3.6. • When opName is specified, the specified opName is the same as an opName of another CCOpSpec within the same CCCmdSpec instance.		
WM23*	9.3.5.1	An ALE implementation SHALL recognize the values defined in the following subsections of 9.3.5.1 as valid operands for the CHECK CCOpSpecType.		By Demonstration TCR – W14
WM24*	9.3.5.1.1	When the fieldspec is epcBank (EPC/UII memory bank), CHECK dataSpec values of the following forms SHALL be recognized: urn:epcglobal:ale:check:iso15962		By Demonstration TCR – W14
WM25*	9.3.5.1.1	When interacting with a Gen2 Tag, an ALE implementation SHALL check the EPC/UII memory bank (Bank 01) of the Tag as follows. A CCOpStatus of MEMORY_CHECK_ERROR SHALL be indicated if any of the following are true: • The toggle bit (bit 17h) is equal to zero. • The AFI bits (bits 18h-1Fh) do not contain an ISO 15962 Application Family Identifier (AFI) that is		By Demonstration TCR – W14

		recognized by the implementation. • The memory bank does not contain an ISO 15962 Data Storage Format Identifier (DSFID) that is recognized by the implementation. • The remaining contents of the memory bank are not valid according to ISO 15962 [ISO15962]. • The remaining contents of the memory bank include two or more data sets having the same object identifier (OID).		
WM26*	9.3.5.1.2	When the fieldspec is userBank (EPC/UII memory bank), CHECK dataSpec values of the following forms SHALL be recognized: urn:epcglobal:ale:check:iso15962		By Demonstration TCR – W14
WM27*	9.3.5.1.2	When interacting with a Gen2 Tag, an ALE implementation SHALL check the User memory bank (Bank 11) of the Tag as follows. A CCOpStatus of MEMORY_CHECK_ERROR SHALL be indicated if any of the following are true: • The memory bank does not contain an ISO 15962 Data Storage Format Identifier (DSFID) that is recognized by the implementation. • The remaining contents of the memory bank are not valid according to ISO 15962 [ISO15962]. • The remaining contents of the memory bank include two or more data sets having the same object identifier (OID).		By Demonstration TCR – W14
WM28*	9.3.5.2	An ALE implementation SHALL recognize the values defined in the following subsections as valid operands for the INITIALIZE CCOpSpecType.		By Demonstration TCR – W14
WM29*	9.3.5.2	An ALE implementation SHALL raise a CCSpecValidationException if the combination of fieldspec and value for the INITIALIZE CCOpSpecType are not recognized.		By Demonstration TCR – W3
WM30*	9.3.5.2.1	When the fieldspec is epcBank (EPC/UII memory bank), INITIALIZE dataSpec values of the following forms SHALL be		By Demonstration TCR – W14

		recognized: urn:epcglobal:ale:init:iso15962:x*AA*[.x*DD*][.force] where *AA* denotes two hexadecimal digits and *DD* denotes two or more hexadecimal digits.		
WM31*	9.3.5.2.1	When interacting with a Gen2 Tag, an ALE implementation SHALL initialize the EPC/UII memory bank (Bank 01) of the Tag as follows: Write a one into bit 17h, write the value *AA* into bits 18h-1Fh, write the value *DD* beginning at bit 20h (the number of bits so written being four times the number of characters in *DD*), followed by eight zero bits (note: the eight zero bits indicate that there are no ISO data sets in the EPC/UII memory bank). Subsequent operations on the Tag will interpret *AA* as the ISO 15962 Application Family Identifier (AFI), and *DD* as the 3203 ISO 15962 Data Storage Format Identifier (DSFID). If x*DD* is omitted, the ALE implementation SHALL supply a default value for *DD*.		By Demonstration TCR – W5, W14
WM32*	9.3.5.2.1	If the optional .force is not present in the dataSpec value, then the ALE implementation SHALL omit all initialization steps as described above if the prior contents of the bits 17h is a one, and the prior contents of bits 18h through 27h are non-zero.		By Demonstration TCR – TCR W5, W14
WM33*	9.3.5.2.1	When interacting with a Gen1 Tag, the implementation SHALL raise an "operation not possible" condition.		N/A
WM34*	9.3.5.2.2	When the fieldspec is userBank (User memory bank), INITIALIZE dataSpec values of the following form SHALL be recognized: urn:epcglobal:ale:init:iso15962:[x*DD*][.force] where *DD* denotes two or more hexadecimal digits.		By Demonstration TCR – TCR W5, W14
WM35*	9.3.5.2.2	When interacting with a Gen2 Tag, 3227 an ALE implementation SHALL initialize the User memory bank (Bank 11) of the Tag as follows: 3229 Write the value *DD* beginning at bit		By Demonstration TCR – TCR W5, W14

		00h (the number of bits so written being four times the number of characters in *DD*), followed by eight zero bits (note: the eight zero bits indicate that there are no ISO data sets in the EPC/UII memory bank). Subsequent operations on the Tag will interpret *DD* as the ISO 15962 Data Storage Format Identifier (DSFID). If x*DD* is omitted, the ALE implementation SHALL supply a default value for *DD*.		
WM36*	9.3.5.2.2	If the optional .force is not present in the dataSpec value, then the ALE implementation SHALL omit all initialization steps as described above if the prior contents of the bits 00h through 07h are non-zero.		By Demonstration TCR – TCR W5, W14
WM37*	9.3.5.2.2	When interacting with a Gen1 Tag, the implementation SHALL raise an "operation not possible" condition.		N/A
WM38	9.3.6	The ALE implementation SHALL interpret the fields of a CCOpDataSpec as given in Table 65.		By Demonstration TCR – W4, W5, W6, W7, W8, W9, W10, W11, W12, W13, W14
WM39	9.3.6	The ALE implementation SHALL use Table 66 to determine what data value is used for the command that includes a CCOpDataSpec		By Demonstration TCR - W5, W10, W11, W12, W13, W14
WM40	9.3.6	The define and immediate methods SHALL raise a CCSpecValidationException if any of the following are true for a CCOpDataSpec instance, according to the value of specType in Table 67. In addition, the define and immediate methods SHALL raise a CCSpecValidationException if a CCOpDataSpec instance is supplied but in Table 64 the opType specifies "[must be omitted]" in the fourth column.		By Demonstration TCR – W3
WM41	9.3.8	The ALE implementation SHALL interpret the data parameter of a LOCK command as given in Table 68.		By Demonstration TCR – W5
WM42	9.3.8	The ALE implementation SHALL interpret "subsequent privileged operations" when interacting with a Gen2 Tag as given in Table 69		By Demonstration TCR – W5, W10

WM43	9.3.10	The define and immediate methods of the ALECC API (Section 9.1) SHALL raise a CCSpecValidationException 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. • The logicalReaders parameter of CCSpec is null, omitted, is an empty list, or contains any logical reader names that are not known to the implementation. • The boundarySpec parameter of CCSpec is null or omitted. • The cmdSpecs parameter of CCSpec is null, omitted, or empty. • The duration, repeatPeriod, or noNewTagsInterval parameter of CCBoundarySpec is negative. • Any element of the startTriggerList or stopTriggerList parameter of CCBoundarySpec 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]. • The tagsProcessedCount of CCBoundarySpec is negative. • No stopping condition apart from afterError is specified 3320 in CCBoundarySpec; *i.e.*, stopTriggerList is empty, and neither duration nor tagsProcessedCount nor noNewTagInterval nor any vendor extension stopping condition is specified. • Any CCCmdSpec instance has a name that is an empty string or that is not accepted by the implementation according to Section 4.5. • Two CCCmdSpec instances have identical values for their name fields. • The patList parameter of any ECFilterListMember instance is		By Demonstration TCR – W3

		empty, null, or omitted, or any element of patList does not conform to the syntax rules for patterns implied by the specified fieldspec. • The opType parameter of a CCOpSpec is not one of the standard opType values specified in Section 9.3.5, or an implementation-specific value known to the ALE implementation. • The opType parameter of a CCOpSpec requires a fieldspec, and fieldspec is null or omitted. • The opType parameter of a CCOpSpec does not require a fieldspec, and fieldspec is specified. • The fieldspec parameter of a CCOpSpec is invalid according to Section 8.2.12. • The opType parameter of a CCOpSpec requires a dataSpec, and dataSpec is null or omitted. • The opType parameter of a CCOpSpec does not require a dataSpec, and dataSpec is specified. • The dataSpec parameter of a CCOpSpec is invalid according to Section 9.3.6. • The dataSpec parameter of a CCOpSpec specifies a value that is invalid for the specified operation, as specified in Section 9.3.6. • Two or more CCOpSpec instances within the same CCCmdSpec instance specify the same (non-empty) opName. • Any value of CCStatProfileName is not recognized, or is recognized but the specified statistics report is not supported.		
WM44	9.4	The implementation SHALL include these fields according to the following definitions in Table 70.		By Demonstration TCR – W4, W5, W6, W7, W8, W9, W10, W11, W12, W13, W14
WM45	9.4.1	The ALE implementation SHALL set the initiationCondition field of a CCReports		By Demonstration TCR – W4, W5,

		instance generated at the conclusion of a command cycle according to the condition that caused the command cycle to start, as specified in Table 71		W6, W7, W8, W9, W10, W11, W12, W13, W14
WM46	9.4.2	The ALE implementation SHALL set the terminationCondition field of a CCReports instance generated at the conclusion of a command cycle according to the condition that caused the command cycle to end, as specified Table 72		By Demonstration TCR – W4, W5, W6, W7, W8, W9, W10, W11, W12, W13, W14
WM47	9.4.3	An ALE implementation SHALL construct a CCCmdReport as given in Table 73.		By Demonstration TCR – W4, W5, W6, W7, W8, W9, W10, W11, W12, W13, W14
WM48	9.4.4	An ALE implementation SHALL construct a CCTagReport as given in Table 74.		By Demonstration TCR – W4, W5, W6, W7, W8, W9, W10, W11, W12, W13, W14
WM49	9.4.5	An ALE implementation SHALL construct a CCOpReport as in Table 75		By Demonstration TCR – W4, W5, W6, W7, W8, W9, W10, W11, W12, W13, W14
WM50	9.4.5	The value of the data field SHALL be constructed according to Table 76.		By Demonstration TCR – W4, W5, W6, W7, W8, W9, W10, W11, W12, W13, W14
WM51	9.4.6	An ALE implementation SHALL return CCStatus codes according to Table 77.		By Demonstration TCR – W4, W5, W6, W7, W8, W9, W10, W11, W12, W13, W14
WM52	9.4.7	An ALE implementation SHALL construct a CCTagStat as given in Table 78.		By Demonstration TCR – W4, W5, W6, W7, W8, W9, W10, W11, W12, W13, W14
WM53	9.5	An ALE implementation SHALL implement the methods of the ALE Writing API for EPCCache as specified in Table 79.		By Demonstration TCR – W11
WM54	9.5	For defineEPCCache, if spec is null, the implementation SHALL use default		By Demonstration TCR – W11

		settings for any controls over the operation fo the EPC Cache. (in Table 79)		
WM55	9.5	The implementation SHALL maintain each defined EPC Cache in the following manner. An EPC Cache is an ordered list of EPCs, whose initial contents is specified by the replenishment argument to defineEPCCache. The EPC Cache may be referred to by name in a CCOpDataSpec whose specType is equal to CACHE. Each time during a command cycle that a Tag is processed using that CCOpDataSpec, the first element of the EPC Cache is removed and used as the value for the operation specified in the CCOpSpec. If there is no first element (because the EPC Cache is empty), then the operation results in an EPC_CACHE_DEPLETED error that is reported in the CCOpReport for that Tag. At any time, the ALE client may add more EPCs to the end of list by invoking the replenishEPCCache method.		By Demonstration TCR – W11
WM56		The EPCCacheSpecValidationException SHALL NOT be raised, however, if the spec argument to defineEPCCache is null, or if the implementation has not made any extensions to EPCCacheSpec. Moreover, all implementations SHALL raise this exception if the specified cacheName is an empty string or is not accepted by the implementation according to Section 4.5.		By Demonstration TCR – W11
WM57	9.5.1	An ALE implementation SHALL raise the appropriate exception listed in Table 81 when the corresponding condition described in Table 80 and in Section 9.1.1 occurs.		By Demonstration TCR – W3
WM58	9.5.3	An ALE implementation SHALL interpret the fields of EPCPatternList as given in Table 82.		By Demonstration TCR - – TCR W5, W14
WM59	9.5.3	An ALE implementation SHALL interpret each EPC pattern URI element of patterns as denoting an ordered list of individual EPCs obtained by enumerating in ascending numerical order all EPCs that match the pattern.		By Demonstration TCR – W11

WM60	9.5.3	An ALE implementation SHALL interpret the overall EPCPatternList instance as denoting an ordered list of individual EPCs obtained by concatenating, in order, the EPCs denoted by each EPC pattern URI element.		By Demonstration TCR – W11
WM61	9.6	An ALE implementation SHALL implement the methods of the ALE Writing API for Association Table as specified in Table 83		By Demonstration TCR – W12
WM62	9.6	The value field of each entry returned by getAssocTableEntries SHALL be in the format specified when the table was defined.		By Demonstration TCR – W12
WM63	9.6.1	An ALE implementation SHALL raise the appropriate exception listed in Table 85 when the corresponding condition described in Table 84 and in Section 9.1.1 occurs.		By Demonstration TCR – W3
WM64	9.6.2	An ALE implementation SHALL interpret an AssocTableSpec instance as in Table 86.		By Demonstration TCR – W12
WM65	9.6.2	The defineAssocTable method SHALL raise an AssocTableValidationException if any of the following are true: • The value of datatype is not a valid datatype as specified in Section 6.2 or a datatype recognized as a vendor extension. • The value of format is not a valid format for the specified datatype.		By Demonstration TCR – W3
WM66	9.6.3	An ALE implementation SHALL interpret the fields of AssocTableEntryList as given in Table 87.		By Demonstration TCR – W12
WM67	9.6.4	An ALE implementation SHALL interpret the fields of AssocTableEntry as given in Table 88.		By Demonstration TCR – W12
WM68	9.7	An ALE implementation SHALL implement the methods of the ALE Writing API for the Random Number Generator as specified in Table 89.		By Demonstration TCR – W13
WM69	9.7.1	All implementations SHALL raise the RNGValidationException if the specified rngName is an empty string or is not accepted by the implementation according to Section 4.5.		By Demonstration TCR – W13

WM70	9.7.1	An ALE implementation SHALL raise the appropriate exception listed in Table 90 when the corresponding condition described in Table 89 and in Section 9.1.1 occurs.		By Demonstration TCR – W3
WM71	9.7.2	Implementations SHALL provide documentation specifying both how the parameters are interpreted by defineRNG and how the parameters are set when returned from getRNG.		By Demonstration TCR – W13 Provide required documentation
WM72	9.7.2	An ALE implementation SHALL interpret an RNGSpec instance as given in Table 92.		By Demontration TCR – W13
WM73	9.7.2	The number of bits for the random numbers generated by this random number generator. Random numbers SHALL be in the range 0 through $2_{length}-1$, inclusive.		By Demonstration TCR – W13
WM74	9.7.2	The defineRNG method SHALL raise an RNGValidationException if length is not a positive integer		By Demonstration TCR – W13
WM75	9.8	Referring to the state transition tables in Section 5.6.1, whenever a transition specifies that "reports are delivered to subscribers" the ALE implementation SHALL attempt to deliver the results to each subscriber by invoking the callbackResults method of the ALECCCallback interface once for each subscriber, passing the CCReports for the command cycle as specified above, and using the binding and addressing information specified by the notification URI for that subscriber as specified in the subscribe call.		By Demonstration TCR – W4, W5, W10

9 ALE 1.1 Tag Memory API Functional Requirements

The ALE 1.1 defines specific functionality that a valid ALE Tag Memory API implementation must provide. The following tables outline the specific requirements that must be tested as defined by the ALE 1.1 specification. Each test requirement entry references the ALE 1.1 Specification and the test case requirement (TCR) used to verify functionality as defined in section 16 of this document.

9.1 Tag Memory API Mandatory Requirements Matrix

The following table outlines the mandatory requirements for an ALE Memroy implementation as defined by the ALE 1.1 Specification. All mandatory Tag Memory API requirements have a requirement number of TMx where x is a decimal number. Any requirement whose requirement number has an asterisk (*) following it is optional and only tested if an implementation has implemented the feature.

Req. No.	Protocol Sub-Clause	Requirements (Requirements, Command, …)	Applies to (ref)	How Verified (by Demonstration or by Design)
TM1	7	An implementation of The Tag Memory Specification API SHALL provide the `TMFixedFieldListSpec` specified in Section 7.3, and SHALL also provide the `TMVariableFieldListSpec` as specified in Section 7.5		By Demonstration TCR – T2, T4
TM2	7.1	An ALE implementation SHALL implement the methods of the ALE Tag Memory Specification API as specified in Table 23		By Demonstration TCR – T1, T2, T3, T4, T5
TM3*	7.1.1	If the Tag Memory API implementation is associated with an implementation of the Access Control API (Section 11), the implementation SHALL raise a security exception if the client was not granted access rights to the called method as specified in Section 11.		By Demonstration TCR – A3
TM4	7.1.1	An ALE implementation SHALL raise the appropriate exception listed in Table 25 when the corresponding condition as described in Table 24 occurs.		By Demonstration TCR – T3
TM5	7.2	An ALE implementation SHALL support `TMFixedFieldListSpec` as a possible type of `TMSpec`.		By Demonstration TCR – T2, T4
TM6	7.2	An ALE implementation also SHALL support `TMVariableFieldListSpec` as a possible type of `TMSpec`.		By Demonstration TCR – T5
TM7	7.2	For all subtypes of `TMSpec`, the `defineTMSpec` method SHALL raise a `TMSpecValidationException` if any of the following are true: • Any component of the specified `TMSpec` attempts to create a fieldname that has previously been defined through the Tag Memory Specification API, or is one of the		By Demonstration TCR – T3

		built-in fieldnames specified in Section 6.1. The latter includes any fieldname that begins with the '@' character.		
TM8	7.4	A TMFixedFieldSpec specifies a single fixed-length field. An ALE implementation SHALL interpret the fields as given in Table 26		By Demonstration TCR – T4
TM9	7.4	The `defineTMSpec` method SHALL raise a `TMSpecValidationException` if any of the following are true: • The value of `fieldname` is a name that has already been defined through the Tag Memory Specification API, or is one of the built-in fieldnames specified in Section 6.1. The latter includes any fieldname that begins with the '@' character. • The value of `fieldname` is the same as the fieldname parameter of another member of the same `TMFixedFieldListSpec`. • The value of `bank` is negative. • The value of `length` is zero or negative. • The value of `offset` is negative. • The value of `defaultDatatype` is not a known datatype, or is not a valid datatype for the specified `bank`, `length`, and `offset` (for example, if the datatype requires more bits than have been provided by `length`). • The value of `defaultFormat` is not a known format, or is not a valid format for the specified `defaultDatatype`.		By Demonstration TCR – T3
TM10	7.6	A TMVariableFieldSpec specifies a variable field This type allows ALE clients to associate a symbolic name with		By Demostration TCR – T5

		an ISO 15962 object identifier. The associated datatype SHALL be `iso-15962-string` and the format SHALL be `string`.		
TM11	7.6	An ALE implementation SHALL interpret TMVariableFieldSpec fields as given in Table 27.		By Demostration TCR – T5
TM12	7.6	The `defineTMSpec` method SHALL raise a `TMSpecValidationException` if any of the following are true: • The value of `fieldname` is a name that has already been defined through the Tag Memory Specification API, or is one of the built-in fieldnames specified in Section 6.1. The latter includes any fieldname that begins with the '@' character. • The value of `fieldname` is the same as the fieldname parameter of another member of the same `TMVariableFieldListSpec`. • The value of `bank` is negative. • The value of `oid` is not valid syntax according to [RFC3061].		By Demonstration TCR – T3

10 ALE 1.1 Access Control API Functional Requirements

The ALE 1.1 defines specific functionality that a valid ALE Access Control API implementation must provide. The following tables outline the specific requirements that must be tested as defined by the ALE 1.1 specification. Each test requirement entry references the ALE 1.1 Specification and the test case requirement (TCR) used to verify functionality as defined in section 17 of this document.

10.1 Access Control API Mandatory Requirements Matrix

The following table outlines the mandatory requirements for an ALE Access Control implementation as defined by the ALE 1.1 Specification. All mandatory access control API requirements have a requirement number of AMx where x is a decimal number. Any requirement whose requirement number has an asterisk (*) following it is optional and only tested if an implementation has implemented the feature.

Req. No.	Protocol Sub-	Requirements (Requirements, Command, …)	Applies to	How Verified (by Demonstration or by

	Clause		(ref)	Design)
AM1	11.1	An ALE implementation SHALL implement the methods of the ALE Access Control API as specified in Table 102.		By Demonstration TCR – A1, A2, A3, A4
AM2	11.2	The implementation SHALL raise the SecurityException if the client was not granted access rights to the called method.		By Demonstration TCR – A3
AM3	11.2	An ALE implementation SHALL raise the appropriate exception listed in Table 104 when the corresponding condition described in Table 103 occurs.		By Demonstration TCR – A4
AM4	11.3	The ALE implementation SHALL interpret the fields of an ACClientIdentity as given in Table 105		By Demonstration TCR – A3
AM5	11.3	The defineClientIdentity, and updateClientIdentity methods of the Access Control API SHALL raise a ClientIdentityValidationException under any of the following circumstances: • One or more of the specified credentials is not a valid credential, according to the implementation-specific rules for validating credentials. • One or more of the specified roleNames is not a known name for a role.		By Demonstration TCR – A4
AM6	11.5	The ALE implementation SHALL interpret the fields of an ACRole as given in Table 106.		By Demonstratoin TCR – A3
AM7	11.5	The defineRole, and updateRole methods of the Access Control API SHALL raise a RoleValidationException under any of the following circumstances: • One or more of the specified permissionNames is not a known name for a permission.		By Demostration TCR – A4
AM8	11.6	The ALE implementation SHALL interpret the fields of an ACPermission as given in Table 107		By Demonstration TCR – A4
AM9	11.6	The definePermission, and updatePermission methods of the Access Control API SHALL raise a PermissionValidationException under any of the following circumstances:		By Demonstration TCR – A4

		• The specified permissionClass is not a known permission class. • One or more of the specified instances is not a valid instance string for the specified permission class, according to the table in Section 11.7.		
AM10	11.7	An ALE implementation SHALL recognize the following permission class names, and implement each according to Table 108.		By Demonstration TCR – A3
AM11	11.7	If a client has not been granted permission for a given method, if that client calls the method the ALE implementation SHALL raise a SecurityException. However, an ALE implementation SHALL NOT raise a SecurityException for a method whose specification does not include SecurityException as a possible error condition, regardless of permission settings. This includes the getStandardVersion and getVendorVersion methods of all ALE APIs, and the getSupportedOperations method of the Access Control API.		By Demonstration TCR – A3
AM12	11.7.1	An ALE implementation SHALL recognize strings in Table 109 as API names when they appear as instances for the Method permission class, denoting that permission is granted to use all methods of the specified API, including vendor extensions.		By Demonstration TCR – A3
AM13	11.7.1	An ALE implementation SHALL recognize a string of that form as a method name when it appears as an instance for the Method permission class.		By Demonstration TCR – A3
AM14	11.7.1	An ALE implementation SHALL recognize the string consisting of a single asterisk character (*) as denoting all methods of all APIs when it appears as an instance for the Method permission class.		By Demonstration TCR – A3
AM15	11.8	An implementation of the Access Control API SHALL implement all methods as specified in Section 11.1.		By Demonstration TCR – A1, A2, A3
AM16	11.8	If an implementation raises UnsupportedOperationException from any Access Control API method, it		By Demonstration TCR – A2 Note: part of the

		SHALL provide documentation that specifies how the client or user controls components of the access control model – client identities, roles, and permissions – for which Access Control API methods raise the UnsupportedOperationException.		demonstration is providing the required documentation.
AM17	11.8	In order to insure that implementations provide a reasonable set of facilities to clients, an ALE implementation SHALL conform to the rules in Section 11.8 (that follow the non-Normative note) for selecting which methods raise UnsupportedOperationException.		By Demonstration TCR – A2
AM18	11.8	An implementation SHALL always support getStandardVersion, getVendorVersion, and getSupportedOperations.		By Demonstration TCR – A2
AM19	11.8	As a consequence of the rules in Section 11.8, the list returned by getSupportedOperations SHALL always include the strings getStandardVersion, getVendorVersion, an getSupportedOperations (and possibly others).		By Demonstration TCR – A2
AM20	11.9	An implementation SHALL provide documentation to specify whether an anonymous client identity is provided, and if so what its name is.		By Demonstration TCR – A2 Provide required documentation
AM21	11.10	In order to grant access to ordinary clients, there must exist at least one client who has permission to use the Access Control API, or there must be some out-of-band mechanism for establishing access permissions. An implementation SHALL provide documentation that specifies how this is done.		By Demonstration TCR – A2 Provide required documentation

11 ALE 1.1 Logical Reader API Functional Requirements

The ALE 1.1 defines specific functionality that a valid ALE Logical Reader API implementation must provide. The following tables outline the specific requirements that must be tested as defined by the ALE 1.1 specification. Each test requirement entry references the ALE 1.1 Specification and the test case requirement (TCR) used to verify functionality as defined in section 18 of this document.

11.1 Logical Reader API Mandatory Requirements Matrix

The following table outlines the mandatory requirements for an ALE Logical Reader API implementation as defined by the ALE 1.1 Specification. All mandatory Logical Reader API requirements have a requirement number of AMx where x is a decimal number. Any requirement whose requirement number has an asterisk (*) following it is optional and only tested if an implementation has implemented the feature.

Req. No.	Protocol Sub-Clause	Requirements (Requirements, Command, …)	Applies to (ref)	How Verified (by Demonstration or by Design)
LM1	10.3	An ALE implementation SHALL implement the methods of the ALE Logical Reader API as specified in Table 93.		By Demonstration TCR – L1, L2, L3, L4, L5, L6
LM2	10.3	getLogicalReaderNames returns an unordered list of the names of all logical readers that are visible to the caller. This list SHALL include both composite readers and base readers.		By Demonstration TCR – L2
LM3	10.3	The setProperties method SHALL modify the properties of a logical reader according to Table 94.		By Demonstration TCR – L4
LM4	10.3	An ALE implementation SHALL provide documentation of what "as soon as possible" means		By Demonstration, TCR – L5 Provide required documentation
LM5	10.3	The update, addReaders, setReaders, removeReaders, and setProperties methods SHALL NOT raise an InUseException if no EC/CCSpecs are active.		By Demonstration TCR – L2, L3, L4, L5
LM6	10.3	The undefine method SHALL raise an InUseException if there exist one or more ECSpecs, CCSpecs, or other LRSpecs that refer to it, whether ECSpecs or CCSpecs are in the active state or not.		By Demonstration TCR – L5
LM7	10.3.1	The ImmutableReaderException SHALL NOT be raised for a composite reader or an API-defined base reader.		By Demonstration TCR – L4
LM8	10.3.1	If the Logical Reader API implementation is associated with an implementation of the Access Control API (Section 11), the Logical Reader API implementation SHALL raise the SecurityException exception if the client was not granted access rights to the called method as specified in Section 11.		By Demonstration TCR – A3

LM9	10.3.1	An ALE implementation SHALL raise the appropriate exception listed In Table 96 when the corresponding condition described in Table 95 occurs.		By Demostration TCR – L4, L5
LM10	10.3.2	An implementation of the Logical Reader API SHALL implement all of the methods defined in Section 10.3. In addition, the following conformance requirements that depend on the type of logical reader apply as given in Table 97.		By Demonstration TCR – L1, L2, L3, L4, L5
LM11	10.4	The ALE implementation SHALL interpret the fields of an LRSpec as given in Table 98.		By Demonstration TCR – L2, L4
LM12	10.4	The define or update methods of the Logical Reader API SHALL raise a ValidationException under any of the following circumstances: • isComposite is false and readers is specified and non-empty. • isComposite is false and the implementation does 3889 not support using the Logical Reader API to define base readers. • isComposite is false, the implementation does support using the Logical Reader API to define base readers, but the LRSpec does not conform to the vendor-specific rules for such use. • isComposite is true and any element of readers is not a known Logical Reader name. • A property name in properties is not recognized by the implementation. • The value specified for a property is not a legal value for that property.		By Demonstration TCR – L5, L6
LM13	10.5	The ALE implementation SHALL interpret the fields of an LRProperty as given in Table 99		By Demonstration TCR – L5
LM14	10.6	The application of this smoothing state machine is that, at any point in time, a Reader SHALL consider a Tag to be within view if the Tag is in the Observed state.		By Demonstration TCR – L4
LM15	10.6	If an ALE implementation supports smoothing (that is, if an ALE		By Demonstration TCR – L4

		implementation does not raise a ValidationException when a client sets the properties defined below), then it SHALL apply the above rule when the reader is used in an ECSpec,		
LM16	10.6	An ALE implementation SHALL interpret the four parameters of the smoothing state machine as given in Table 101		By Demonstration TCR – L4
LM17	10.6	If all four smoothing properties are set to null for a given logical reader, an implementation SHALL NOT use smoothing for that logical reader.		By Demonstration TCR – L4
LM18	10.6	The define, update, and setProperties methods of the Logical Reader API SHALL raise a ValidationException under any of the following circumstances: • If the value of any of the four properties specified above is a non-null string that is not parseable as a non-negative decimal integer numeral. • If the value of any of the four properties specified above is non-null, and the implementation does not support Tag smoothing for the specified logical reader. • If both ObservedTimeThreshold and ObservedCountThreshold are null, and any of the other smoothing parameters is non-null. • If the implementation does not wish to support the combination of the four parameter values that would result from the operation. An implementation that supports smoothing for the specified logical reader SHALL NOT, however, raise a ValidationException for the case where all four parameters are set to null.		By Demonstration TCR – L5

12 Part II: XML and SOAP Binding Requirements

The ALE 1.1 defines XML and SOAP Bindings. The following tables outline the specific requirements that must be tested as defined by the ALE 1.1 specification. Each

453 test requirement entry references the ALE 1.1 Specification and the test case requirement
454 (TCR) used to verify functionality as defined in sections 14 to 18 of this document.

455 12.1 XML and SOAP Binding Mandatory Requirements Matrix

456 The following table outlines the mandatory requirements for the XML and SOAP
457 bindings as defined by the ALE 1.1 Specification. All mandatory XML and SOAP
458 Binding requirements have a requirement number of XMx where x is a decimal number.
459

Req. No.	Protocol Sub-Clause	Requirements (Requirements, Command, …)	Applies to (ref)	How Verified (by Demonstration or by Design)
XM1*	3.2.1 Part 2	Vendor-specific attributes may be added to any XSD type corresponding to a UML class in which `<<extension point>>` occurs. Vendor-specific attributes SHALL NOT be in the EPCglobal ALE namespace (urn:epcglobal:ale:xsd:1).		By Demonstration TCR-R21, W16, T6, A5, L7
XM2*	3.2.1 Part 2	Vendor-specific attributes SHALL be in a namespace whose namespace URI has the vendor as the owning authority.		By Design TCR-R21, W16, T6, A5, L7
XM3*	3.2.1 Part 2	Declarations of vendor-specific attributes SHALL specify `use="optional"`.		By Demonstration TCR-R21, W16, T6, A5, L7
XM4*	3.2.1 Part 2	Vendor-specific elements SHALL NOT be in the EPCglobal ALE namespace (`urn:epcglobal:ale:xsd:1`)		By Demonstration TCR-R21, W16, T6, A5, L7
XM5*	3.2.1 Part 2	Vendor-specific elements SHALL be in a namespace whose namespace URI has the vendor as the owning authority		By Design TCR-R21, W16, T6, A5, L7

XM6*	3.2.1 Part 2	`<xsd:group name="VendorExtension">`
		`<xsd:sequence>`
		`<!--`
		`Definitions or references to vendor elements go here. Each SHALL specify minOccurs="0".`
		`-->`
				By Demonstration TCR-R21, W16, T6, A5, L7

XM7*	3.2.1 Part 2	Standard attributes may be added to any XSD type corresponding to a UML class in which `<<extension point>>` occurs. Standard attributes SHALL NOT be in any namespace, and SHALL NOT conflict with any existing standard attribute name.		N/A - This requirement only applies to future version of the standard
XM8*	3.2.2	A vendor implementation MAY add		By Demonstration

	Part 2	additional methods to an ALE API, provided that the name of a vendor extension method SHALL NOT conflict with existing methods.		TCR-R21, W16, T6, A5, L7
XM9*	3.2.3 Part 2	Vendor extension values SHALL take the form of absolute URIs [URI], where the URI has the vendor as the owning authority.		By Demonstration TCR-R21, W16, T6, A5, L7
XM10	3.5 Part 2	<xsd:complexType name="ECReportGroup"> … </xsd:sequence> <!-- The groupName attribute SHALL be omitted to indicate the default group. --> <xsd:attribute name="groupName" type="xsd:string" use="optional"/> <xsd:anyAttribute processContents="lax"/> </xsd:complexType>		By Demostration TCR – R5
XM11	3.5 Part 2	<xsd:complexType name="ECReportGroupListMember"> <xsd:sequence> <!-- Each of the following four elements SHALL be omitted if null. --> <xsd:element name="epc" type="epcglobal:EPC" minOccurs="0"/> <xsd:element name="tag" type="epcglobal:EPC" minOccurs="0"/> <xsd:element name="rawHex" type="epcglobal:EPC" minOccurs="0"/> <xsd:element name="rawDecimal" type="epcglobal:EPC" minOccurs="0"/>		By Demonstration TCR – R5
XM12*	5 Part 2	If an implementation provides an additional binding of the callback interface, it SHALL use a URI scheme that does not conflict with any of the standardized bindings.		By Demonstration TCR-R21, W16, T6, A5, L7
XM13	5 Part 2	All notification URIs recognized by bindings as legal, whether the binding is standardized as a part of this specification or not, SHALL conform to the general syntax for URIs as defined in [RFC2396].		By Demonstration TCR – R4
XM14	5.1 Part 2	The interpretation by the ALE implementation of the response code returned by the callback receiver is		By Demonstration TCR – R4

		outside the scope of this specification; however, all implementations SHALL interpret a response code 2xx (that is, any response code between 200 and 299, inclusive) as a normal response, not indicative of any error.		
XM15	5.2 Part 2	The ALE implementation delivers an event cycle or command cycle report by opening a new TCP connection to the specified host and port, writing to the connection the `ECReports` instance or `CCReports` instance encoded in XML according to the schema specified in Section 3.5 or Section 3.6, respectively, and then closing the connection. The ALE implementation SHALL NOT require a reply or acknowledgement.		By Demonstration TCR – W4
XM16	5.4 Part 2	The ALE implementation SHALL deliver event cycle or command 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` (as defined in [RFC2616]), and where the payload is the `ECReports` or `CCReports` instance encoded in XML according to the schema specified in Section 0 or Section 3.6, respectively.		By Demonstration TCR – W4
XM17	5.4 Part 2	For these bindings, HTTP SHALL be used over TLS as defined in [RFC2818]. TLS for this purpose SHALL be implemented as defined in [RFC2246] except that the mandatory cipher suite is `TLS RSA WITH AES 128 CBC SHA`, as defined in [RFC3268] with CompressionMethod.null.		By Demonstrations TCR – W4
XM18	5.4 Part 2	The interpretation by the ALE implementation of the response code returned by the callback receiver is outside the scope of this specification; however, all implementations SHALL interpret a response code 2xx (that is, any response code between 200 and 299, inclusive) as a normal response,		By Demonstration TCR – W4

		not indicative of any error.		

13 Notes on Test Case Requirements

An ALE Conformance Certification Program will test an Implementation Under Test (IUT) according to predefined test case requirements that have been designed to isolate and test specific features and functions of the ALE 1.1 Specification. While these test case requirements are not exhaustive, they test all the mandatory features that are required by the specification.

13.1 Nomenclature

The following nomenclature is used for assigning IDs to the ALE1.1 Conformance Test Cases.

- ALE1.1 Reading API Tests: TCR-R*X*
- ALE1.1 Writing API Tests: TCR-W*X*
- ALE1.1 Tag Memory API Tests: TCR-T*X*
- ALE1.1 Access Control API Tests: TCR-A*X*
- ALE1.1 Logical Reader API Tests: TCR-L*X*

Where *TCR* stands for *"Test Case Requirement"* and *X* = 1, 2, 3…

13.2 General Requirments

There is a requirements matrix for ALE 1.1 general requirements. These are requirments that are defined outside of the API sections of the specification. The general requirements apply to one or more of the APIs specified and will be tested as part of the API test cases.

13.3 Pre-Condidtions and Post-Conditions

Each test has zero or more pre-test conditions defined. In most pre-test conditions the class variables relevant for the corresponding APIs are provided. However, not all elements of all classes are present in the pre-test conditions. Such variables should be considered as either Empty / Null or they have no impact for the corresponding test.

The post-test conditions are omitted in many cases for simplicity. However, care should be taken to remove any undeleted Specs (ECSpec / CCSpec / TMSpec / LRSpec / ACSpec) at the end of the test during the preparation of the testscripts.

13.4 XML Instance Document Validation

For all test case requirments where and XML instance document (e.g. ECReports, CCReports) is returned by the implementation under test, it should be validated against a modified standard ALE 1.1 XSD for that document. A modified ALE 1.1 XSD should be used in which the wildcard has been removed from all inner <extension> elements. This

would ensure that vendors aren't adding extensions into areas that are reserved for new features in future ALE versions.

The vendor XSDs should also be examined to ensure extensions, elements and attributes were not added in places not allowed by the specification. This will require that a vendor make their XSDs available to the certification test lab for inspection.

14 Reading API Test Case Requirements

The ALE 1.1 Reading API Test Case Requirements are provided in the following subsections

14.1 TCR-R1 – Get Version, Reading API

Get Version, Reading API		
TPId: TCR-R1		
Requirement Purpose: This Test Case confirms the proper functions of the ALE methods of the Reading API that return the ALE standard version and the vendor version for the ALE implementation under test. The return of correct version numbers also confirms the correct implementation is being tested. **Requirements Tested:** GM1, GM2, GM3, GM4, GM5, RM1		
Pre-test conditions: • None		
Step	**Step description**	**Expected results**
1	Invoke the getStandardVersion method of the Reading API	• Confirm the string "1.1" is returned. • Confirm the result returned by this method only pertain to the Reading API.
2	Invoke the getVendorVersion method of the Reading API	• Confirm that either an empty string or a string conforming to a proper URI is returned. • Confirm the vendor is the owning authority of the URI if the returned string is not empty (by Design) • Confirm the result returned by this method only pertain to the API to the Reading API.

14.2 TCR-R2 – Defining, Un-defining and Retrieving ECSpecs, Reading API

Defining, Un-defining and Retrieving ECSpecs, Reading API
TPId: TCR-R2
Requirement Purpose: This Test Case confirms that a valid ECSpec can be defined and undefined. Further the defining and un-defining of the ECSpec can be verified with ALE API methods getECSpec and getECSpecNames. **Requirements Tested:** GM1, GM6, GM7, GM12, RM1, RM2 RM5

Pre-test conditions: • No ECSpecs are defined. • Ensure all specName parameters accept as a name any non-empty string of Unicode characters that does not include Pattern_White_Space or Pattern_Syntax characters (see GM6) • The Writing API must be supported for Step 7. Otherwise, Step 7 is optional.		
Step	**Step description**	**Expected results**
1	Invoke the define method with a valid ECSpec without extensions	The ALE implementation contains the ECSpec definition supplied in the define method. Steps 2 and 3 confirm the defining of the ECSpec.
2	Verify the ECSpec was defined by invoking the getECSpecNames method	Verify that the name returned in the list is equivalent to the ECSpec just defined
3	Invoke getECSpec using the name of the defined ECSpec.	Verify that the ECSpec returned is the same as the one defined
4	Invoke undefine to remove the ECSpec that was defined.	The ALE implementation should no longer have the ECSpec defined. Confirmed by step 5.
5	Verify that the ECSpec is undefined by invoking the getECSpecNames method.	Verify that the list returned is empty.
6	Repeat steps 1 through 5 for a valid ECSpec with extensions.	Note that when Step 3 is repeated, the ECSpec returned by getECSpec may not necessarily include any of the extension elements provided in Step 1, if those extensions are not understood by the implementation.
7	Invoke the Writing API define method with a CCSpec and specName = “foo”. Invoke the Reading API define method with an ECSpec and specName = “foo”. (optional, unless the Writing API is also supported)	Verify that the ALE implementation does accept both the CCSpec and the ECSpec and does not raise a DuplicateNameException.

512

513 14.3 TCR-R3 – Exceptions, Reading API

Exceptions, Reading API		
TPId: TCR-R3		
Requirement Purpose: This Test Case confirms that the ALE implementation will raise all exceptions as defined in the ALE specification. This covers exceptions raised due to incorrect parameters passed in ALE API methods and exceptions raised due to missing or invalid parameters in an ECSpec. **Requirements Tested:** GM1, GM6, GM8, GM32, GM38, GM42, GM46, GM50, GM55, GM60, RM62, GM64, GM83, RM1, RM4, RM5, RM7, R12, RM15, RM16, RM20, RM24, RM28, RM31, RM33, RM42, RM46		
Pre-test conditions: • No ECSpecs are defined • Note: The ECSpecs used in this Test Case Requirement steps should be valid except for the conditions specified in step being performed.		
Step	**Step description**	**Expected results**
1	Invoke the getECSpec with an unknown spec name.	Verify that the ALE implementation raises a NoSuchNameException.

2	Invoke the poll method using an unknown name for the ECSpec string.	Verify that the ALE implementation raises a NoSuchNameException is raised.
3	Invoke the subscribe method with an unknown ECSpec name	Verify that the ALE implementation raises a NoSuchNameException.
4	Invoke the unsubscribe method with a defined ECSpec name and a well-formed notification URI. The notification URI should not belong to a user who is subscribed	Verify that the ALE implementation raises a NoSuchSubscriberException.
5	Invoke the subscribe method using the name of a valid and defined ECSpec and a well formed notification URI that is not supported by the implementation under test	Verify that the ALE Implementation raises an InvalidURIException
6	Invoke the unsubscribe method with an unknown ECSpec name	Verify that the ALE implementation raises a NoSuchNameException.
7	Invoke the getSubscribers method with an unknown ECSpec name	Verify that the ALE implementation raises a NoSuchNameException.
8	Invoke the undefine method with an unknown ECSpec name.	Verify that the ALE implementation raises a NoSuchNameException.
9	Invoke the subscribe method with an un-conforming URI	Verify that the ALE Implementation raises an InvalidURIException
10	Invoke the unsubscribe method with an un-conforming URI	Verify that the ALE Implementation raises an InvalidURIException or NoSuchSubscriberException
11	Invoke the define method with a valid ECSpec	The ALE implementation holds the ECSpec definition
12	Verify the ECSpec was defined by invoking a getECSpecNames method	Verify that the name returned is that of the ECSpec just defined
13	Invoke the define method again with a valid ECSpec and the name of the ECSpec defined in step 11.	Verify that the ALE implementation raises a DuplicateNameException
14	The same subscriber should subscribe to the an ECSpec to which the subscriber is already subscribed.	Verify that the ALE implementation raises a DuplicateSubscriptionException
15	Invoke the immediate method with an ECSpec that has a readers parameter that is null, omitted, is an empty list, or contains names that are unknown to the ALE implementation.	Verify that the ALE implementation raises an ECSpecValidationException
16	Invoke the define method with an ECSpec that has a boundaries parameter that is null or omitted.	Verify that the ALE implementation raises an ECSpecValidationException
17	Invoke the immediate method with an ECSpec that has a reportSpecs parameter that is null, omitted, is an empty list or contains two ECReportSpec instances with the same reportName.	Verify that the ALE implementation raises an ECSpecValidationException

18	Invoke the define method with an ECSpec whose ECBoundarySpec contains a duration, repeatPeriod, or stableSetInterval parameter that is negative	Verify that the ALE implementation raises an ECSpecValidationException
19	Invoke the define method with an ECSpec whose ECBoundarySpec has no termination condition specified.	Verify that the ALE implementation raises an ECSpecValidationException
20	Invoke the immediate method with an ECSpec whose ECBoundarySpec has a startTrigger or a stopTrigger which has a value that does not conform to the URI syntax.	Verify that the ALE implementation raises an ECSpecValidationException
21	Invoke the define method with an ECSpec whose ECReportSpecs list contains an ECReportSpec whose filter parameter contains a URI that does not conform to the EPC pattern syntax.	Verify that the ALE implementation raises an ECSpecValidationException
22	Invoke the define method with an ECSpec whose ECReportSpecs list contains a ECReportSpec whose group parameter does not conform to the syntax for grouping patterns in section 6.2.1.3 or contains two grouping patterns that are non-disjoint as defined in section 6.2.1.3.	Verify that the ALE implementation raises an ECSpecValidationException
23	Invoke the immediate method with an ECSpec whose ECReportSpecs list contains an ECReportSpec whose output parameter's ECReportOutputSpec Boolean values for all tag formats (includeEPC, includeTag, includeRawHex, includeRawDecimal and includeCount) are set to false.	Verify that the ALE implementation raises an ECSpecValidationException
24	Invoke the define method with an ECSpec Whose primaryKeyFields contain an Unknown fieldname.	Verify that the ALE implementation raises an ECSpecValidationException
25	Invoke the define method with an ECSpec Whose statProfileNames contain an Unknown element	Verify that the ALE implementation raises an ECSpecValidationException
26	Invoke the define method with an ECSpec Whose ECFilterSpec has filterList with empty patList	Verify that the ALE implementation raises an ECSpecValidationException
27	Invoke the define method with an ECSpec Whose ECFilterSpec has filterList with patList that does not conform syntax rules for patterns	Verify that the ALE implementation raises an ECSpecValidationException
28	Invoke the define method with an ECSpec Whose ECFilterSpec has filterList with fieldspec with unknown datatype and format	Verify that the ALE implementation raises an ECSpecValidationException

29	Invoke the define method with an ECSpec Whose ECgroupSpec has fieldspec with unknown datatype and format	Verify that the ALE implementation raises an ECSpecValidationException
30	Invoke the define method with an ECSpec Whose ECgroupSpec has patternList that does not conform to the syntax rules for grouping patterns	Verify that the ALE implementation raises an ECSpecValidationException
31	Invoke the define method with an ECSpec Whose ECgroupSpec has patternList of non disjoint pattern	Verify that the ALE implementation raises an ECSpecValidationException
32	Invoke the immediate method with an ECSpec whose ECBoundarySpec has a startTriggerList (containing a startTrigger) or a stopTriggerList (containing a stopTrigger). The elements of the startTriggerList and stopTriggerList does not conform to the URI syntax.	Verify that the ALE implementation raises an ECSpecValidationException
33	Invoke the define method with a specName that uses a diacritical letter (e.g. embarcadère). Then invoke the undefine method with a specName that looks equivalent but does not contain the diacritical mark (e.g. embarcadere)	Verify that the ALE implementation raises a NoSuchNameException for the undefined method.
34	Invoke the define method with an ECSpec with a primaryKeyFields whose implementation does not support the primaryKeyFields value with the specificed logical readers.	Verify that the ALE implementation raises an ECSpecValidationException
35	Invoke the define method with a fieldspec that specifies a fieldname of `epc` and specifies a datatype that is not an `epc`.	Verify that the ALE implementation raises an ECSpecValidationException
36	Invoke the define method with a fieldspec that specifies a fieldname beginning with an `@` character but not conforming to any syntax specified in Section 6.1.9 of the specification.	Verify that the ALE implementation raises an ECSpecValidationException

514

515 14.4 TCR-R4 – Subscribe and Unsubscribe, Reading API

Subscribe and Unsubscribe, Reading API
TPId: TCR-R4
Requirement Purpose: This Test Case confirms that clients can subscribe and unsubscribe to ECSpecs that have been correctly defined and the notification URIs used conform to the ALE standard. Multiple subscriptions to the same ECSpec are tested. **Requirements Tested:** GM1, GM13, GM14, GM15, GM16, GM18, RM1, RM64, XM13, XM14

Pre-test conditions: • A valid ECSpec has been defined. • Note: Implementation of only one notification API type is required: HTTP, HTTPS or TCP. All may be certified.		
Step	**Step description**	**Expected results**
1	The first subscriber invokes the subscribe method to subscribe to the ECSpec providing its HTTP Notification URI	A user is subscribed to the ECSpec in the subscribe invocation and the associated event cycle is activated. Step 2 provides verification.
2	Invoke the getSubscribers method to verify that the subscribe method succeeded.	getSubscriber returns the list of notification URIs. The notification URI from step 1 should be in the list. Verify that the correct ECReports are being received at the notification URI per the boundary condition specified in the ECSpec
3	A second subscriber invokes the subscribe method to subscribe to the same ECSpec as step 1 therefore providing its HTTP Notification URI.	A second user is subscribed to the ECSpec in the subscribe invocation. Step 4 provides verification.
4	Invoke the getSubscriber method to verify that the subscribe method succeeded.	getSubscriber returns the list of notification URIs. The notification URI from step 1 and step 3 should be in the list.
5	The first subscriber un-subscribes by invoking the unsubscribe method using its HTTP Notification URI as a parameter.	The first user is unsubscribed.
6	Invoke the getSubscribers method to verify that the unsubscribe method succeeded and the first subscriber is no longer subscribed.	getSubscriber returns the list of notification URIs. The notification URI from step 1 should no longer be in the list.
7	The second subscriber un-subscribes by invoking the unsubscribe method.	The second user is unsubscribed
8	Invoke the getSubscribers method to verify that the unsubscribe method succeeded and the second subscriber is no longer subscribed.	getSubscriber returns the list of notification URIs. The notification URI from step 3 should no longer be in the list.
9	Repeat steps 1 through 8 replacing the HTTP Notification URIs with a TCP Notification URIs	
10	Repeat steps 1 through 8 replacing the HTTP Notification URIs with a File Notification URIs	
11	Repeat steps 1 through 8 replacing the HTTP Notification URIs with a HTTPS Notification URIs	

516

517 14.5 TCR-R5 – Poll, Reading API

Poll, Reading API
TPId: TCR-R5

Requirement Purpose: This Test Case confirms that the invocation of poll method can provide a valid ECSpecName to an ALE implementation and return an ECReports consistent with the parameters set in the ECSpec and within the boundary conditions.
Requirements Tested: GM1, GM9, GM13, GM14, GM15, GM16, GM18, RM1, RM5, RM10, RM11, RM13, RM14, RM19, RM21, RM23, RM25, RM37, RM47, RM48, RM49, RM50, RM51, RM52, RM53, RM56, RM57, RM58, RM63, XM10, XM11

Pre-test conditions:

- Two Valid ECSpecs have been defined: A and B
- Spec A: A valid ECSpec has been defined with a repeatPeriod of M seconds, a duration of N seconds where M > N
- Spec B: A valid ECSpec has been defined with a repeatPeriod of M seconds, a duration of N seconds where M < N as shown below:

ECSpec					
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
Reader List	1 Reader			startTrigger	Null
stopTrigger	Null	startTriggerList	Null	stopTriggerList	Null
duration	N Sec	stableSetInterval	0	repeatPeriod	M sec
reportSet	CURRENT	primaryKeyFields	Null	whenDataAvailable	False
includeCount	No	includeEPC	Yes	includeTag	No
reportIfEmpty	False	reportOnlyOnChange	False	includeSpecInReports	False
includeRawHex	No	includeRawDecimal	No	groupSpec	No
includePatterns	No	excludePatterns	No	filterList	No
statProfileNames	No	reportSpecs	Note 1		

Note 1: The reportsSpec should have at least one name in its list. That name string should be equal to the name string for the ECSpec (tests requirement GM9)

Step	Step description	Expected results
1	Place a tag set in the reader field.	
2	Invoke the poll using the name of spec A (M > N)	After N seconds, an ECReports that conforms to the ALE XSD should be returned listing all the tags in the reader field.
3	Remove all the tags from the reader field.	
4	Invoke the poll using the name of spec A (M > N)	After N seconds, an ECReports that conforms to the ALE XSD and that contains zero ECReport instances should be returned even though reportIfEmpty is false.
5	Place a tag set in the reader field.	
6	Invoke the poll using the name of spec B (M < N)	After N seconds, an ECReports that conforms to the ALE XSD should be returned listing all the tags in the reader field.
7	Remove all the tags from the reader field.	

8	Invoke the poll using the name of spec B (M < N)	After N seconds, an ECReports that conforms to the ALE XSD and that contains zero ECReport instances should be returned even though reportIfEmpty is false.

518

519 ## 14.6 TCR-R6 – Immediate and ECStatProfileName, Reading API

Immediate, Reading API

TPId: TCR-R6

Requirement Purpose: This Test Case confirms that the invocation of the immediate method can provide a valid ECSpec to an ALE implementation and return ECReports consistent with the parameters set in the ECSpec and within the boundary conditions. The test also verifies the inclusion of an ECSpec in and ECReports. This test also optionally verifies ECStatProfilename feature.
Requirements Tested: GM1, GM21, GM22, GM23, GM24, GM25, RM1, RM5, RM6, RM10, RM11, RM19, RM21, RM23, RM25, RM39, RM41, RM43, RM45, RM47, RM48, RM49, RM50, RM51, RM52, RM53, RM56, RM57, RM58, RM59, RM60, RM61, RM62

Pre-test conditions:

- None
- ECSpec for the test:

ECSpec					
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
Reader List	1 Reader	repeatPeriod	M sec	startTrigger, startTriggerList	Null
duration	0	stableSetInterval	N sec	stopTrigger, stopTriggerList	Null
reportSet	ADDITIONS	primaryKeyFields	Null	whenDataAvailable	False
includeCount	No	includeEPC	No	includeTag	Yes
reportIfEmpty	False	reportOnlyOnChange	False	includeSpecInReports	True
includeRawHex	Yes	includeRawDecimal	No	groupSpec	No
includePatterns	No	excludePatterns	No	filterList	No
statProfileNames	No				

Step	Step description	Expected results
1	Place a tag set in the reader field. Keeps the tags in the reader field for a time greater than the N second stableSetInterval once the immediate method is issued.	

2	Invoke the immediate method using a valid ECSpec as specified in the pre-test conditions.	Ensure no ECReports is returned prior to N seconds. Ensure an ECReports that conforms to the ALE XSD is returned immediately after N seconds has past. Confirm the ECReports contains tag EPCs in Tag and RawHex formats and the ECSpec appears in the ECReports.
3	(optional) Repeat steps 1 and 2 with statProfileNames containing "TagTimeStamps".	Result is same as in step 2 plus the time ECTagTimestampStat reported for each tag. The stat timestamps should be compared with the date field of the ECReports to see that the times are synchronized.

520

521 14.7 TCR-R7 – Using startTrigger, startTriggersList, stopTrigger
522 and stopTriggersList, Reading API

523

Using startTrigger, startTriggersList, stopTrigger and stopTriggersList, Reading API

TPId: TCR-R7

Requirement Purpose: This Test Case confirms the following features of the ECSpec: startTrigger, startTriggerList, stopTrigger and stopTriggerList.
Requirements Tested: GM1, RM6, RM10, RM11, RM19, RM21, RM23, RM25, RM26, RM39, RM40, RM41, RM43, RM47, RM48, RM49, RM50, RM51, RM52, RM53, RM56, RM57, RM58, RM64

Pre-test conditions:

- A valid ECSpec has been defined as shown.

ECSpec					
Parameter	Value	Parameter	Value	Parameter	Value
Reader List	1 Reader	repeatPeriod	0	startTrigger	Yes
duration	0	stableSetInterval	0	stopTrigger	Yes
reportSet	CURRENT	whenDataAvailable	False	startTriggerList	Yes
includeCount	Yes	includeEPC	No	stopTriggerList	Yes
reportIfEmpty	False	reportOnlyOnChange	False	primaryKeyFields	Null
includeRawHex	No	includeRawDecimal	Yes	includeTag	Yes
includePatterns	No	excludePatterns	No	includeSpecInReports	False
filterList	No	statProfileNames	No	groupSpec	No

- There are no users subscribed to the ECSpec.
- StartTriggerList contains startTriggers URI1 and URI2. StopTriggerList contains stopTriggers URI3 and URI4.
- The startTrigger is URI5 and the stopTrigger is URI6.

Step	Step description	Expected results
1	Invoke the poll method to activate the ECSpec	None

2	Move a set of tags into the reader field and trip the start trigger URI1.	The Event Cycle should begin
3	Trip the stop trigger URI3 after a sufficient time has passed for all the tags to have been read and reported to the ALE implementation.	An ECReports that conforms to the ALE XSD should be retuned by the poll command issued in step 1. The ECReports should contain an ECReport that contains the tag EPCs of those tags place in the reader field. The ECReports should also contain startTrigger URI1 as initiationTrigger and stopTrigger URI3 as terminationTrigger.
4	Invoke the poll method to activate the ECSpec	None
5	Move a set of tags into the reader field and trip the start trigger URI2.	The Event Cycle should begin
6	Trip the stop trigger URI4 after a sufficient time has passed for all the tags to have been read and reported to the ALE implementation.	An ECReports that conforms to the ALE XSD should be retuned by the poll command issued in step 1. The ECReports should contain an ECReport that contains the tag EPCs of those tags place in the reader field. The ECReports should also contain startTrigger URI2 as initiationTrigger and stopTrigger URI4 as terminationTrigger.
7	Invoke the poll method to activate the ECSpec	None
8	Move a set of tags into the reader field and trip the start trigger URI5.	The Event Cycle should begin
9	Trip the stop trigger URI6 after a sufficient time has passed for all the tags to have been read and reported to the ALE implementation.	An ECReports that conforms to the ALE XSD should be retuned by the poll command issued in step 1. The ECReports should contain an ECReport that contains the tag EPCs of those tags place in the reader field. The ECReports should also contain startTrigger URI5 as initiationTrigger and stopTrigger URI6 as terminationTrigger.

524
525

526 14.8 TCR-R8 – Exclude Filtering, Reading API

527

Exclude Filtering, Reading API
TPId: TCR-R8
Requirement Purpose: This Test Case Requirement confirms the following features of the ECSpec: include count, include current, includeTag format, includeRawDecimal format, reportIfEmpty=false, exclude pattern filtering and one logical reader in the reader list. It also tests for support of the built-in fieldnames. **Requirements Tested:** GM1, GM26, GM27, GM28, GM29, GM33, GM34, GM35, GM36, GM37, GM38, GM39, GM40, GM42, GM43, GM44, GM45, GM46, GM47. GM48, GM49, GM50, GM51, GM52, GM54, GM55, GM56, GM57, GM58, GM59, GM60, GM67, GM69, GM71, GM82, GM84, GM85, GM86, GM87, GM88, GM89, GM90, GM91, GM93, GM94, GM96, GM97, GM98, GM99, GM100, GM101, GM102, GM103, GM104, GM105, GM106, GM107, GM108, GM109, GM110, GM111, GM112, RM1, RM19, RM21, RM23, RM25, RM39, RM40, RM41, RM43

Pre-test conditions:

- A valid ECSpec has been defined as shown. The repeatPeriod = the duration (M = N)

ECSpec					
Parameter	Value	Parameter	Value	Parameter	Value
Reader List	1 Reader	repeatPeriod	M sec	startTrigger, startTriggerList	omitted
duration	N sec	stableSetInterval	0	stopTrigger, stopTriggerList	omitted
reportSet	CURRENT	whenDataAvailable	False		
includeCount	Yes	includeEPC	No	includeTag	Yes
reportIfEmpty	False	reportOnlyOnChange	False	includeSpecInReports	False
includeRawHex	No	includeRawDecimal	Yes	groupSpec	No
includePatterns	No	excludePatterns	No	filterList	Yes
statProfileNames	No				

ECFilterListMember					
Parameter	Value	Parameter	Value	Parameter	Value
includeExclude	Exclude	Fieldspec	ECFieldspec1/ ECFieldspec2/ ECFieldspec3/ ECFieldspec4	patList	*valid pattern list*

ECFieldSpec1					
Parameter	Value	Parameter	Value	Parameter	Value
Fieldname	epc	Datatype	epc	Format	epc-tag

ECFieldSpec2					
Parameter	Value	Parameter	Value	Parameter	Value
Fieldname	killPwd	Datatype	omitted	Format	omitted

ECFieldSpec3					
Parameter	Value	Parameter	Value	Parameter	Value
Fieldname	accessPwd	Datatype	omitted	Format	omitted

ECFieldSpec4					
Parameter	Value	Parameter	Value	Parameter	Value
Fieldname	afi	Datatype	omitted	Format	omitted

ECFieldSpec5					
Parameter	Value	Parameter	Value	Parameter	Value
Fieldname	nsi	Datatype	omitted	Format	omitted

- There are no users subscribed to the ECSpec.

Step	Step description	Expected results
1	Invoke the poll method to activate the ECSpec (using **ECFieldSpec1**) and begin the event cycle. Ensure that a Gen2 tag set is in the reader field.	The event cycle should have started.
2	Verify that after time M, when the repeatPeriod expires, an ECReports is returned by the poll.	An ECReports that conforms to the ALE XSD should be returned by the poll in Step 1. It should include those tags from the tag set that did not match the exclude filter. The tag identities should be provided in Tag and Raw Decimal Format and should be consistent with the EPCs on the tags. A count of the tags should be in the report.
3	Invoke the poll method again but only have tags in the tag set presented to the reader field where all Gen2 tags that will be filtered out by exclude filter.	After the repeatPeriod expires after time M, an ECReports that conforms to the ALE XSD should be returned. It should include an empty ECReports.
4	Repeat steps 1-3 with **ECFieldSpec2**	Verify that the tags with the same killpwd value are excluded from the ECReport.
5	Repeat steps 1-3 with **ECFieldSpec3**	Verify that the tags with the same accesspwd value are excluded from the ECReport.
6	Repeat steps 1-3 with **ECFieldSpec4**	Verify that the tags with the same afi value are excluded from the ECReport.
7	Repeat steps 1-3 with **ECFieldSpec5**	Verify that the tags with the same nsi value are excluded from the ECReport.
8	Repeat steps 1-3 with **ECFieldSpec2** with tags or readers that don't support killPwd	Verify that the tags with the same killpwd value are excluded from the ECReport. The "operation not possible" should be raised.
9	Repeat steps 1-3 with **ECFieldSpec3** with tags or readers that don't support accessPwd	Verify that the tags with the same accesspwd value are excluded from the ECReport. The "operation not possible" should be raised.
10	Repeat steps 1-3 with **ECFieldSpec4** with tags or readers that don't support afi.	Verify that the tags with the same afi value are excluded from the ECReport. The "operation not possible" should be raised.
11	Repeat steps 1-3 with **ECFieldSpec5** with tags or readers that don't support nsi.	Verify that the tags with the same nsi value are excluded from the ECReport. The "operation not possible" should be raised.

528

529 14.9 TCR-R9 – Using whenDataAvailable, Reading API

Using whenDataAvailable, Reading API
TPId: TCR-R9

Requirement Purpose: This Test Case confirms the correct operation of the feature whenDataAvailable of the ECSpec. Also, in this test includePatterns, excludePattern and filterList are verified.
Requirements Tested: GM1, RM1, RM10, RM11, RM19, RM21, RM25, RM26, RM64

Pre-test conditions:

- Two Valid ECSpecs have been defined: A and B
- Spec A: A valid ECSpec has been defined with a repeatPeriod of M seconds, a duration of N seconds where M > N and whenDataAvailable = true
- Spec B: A valid ECSpec has been defined with a repeatPeriod of M seconds, a duration of N seconds where M > N and whenDataAvailable = false as shown below:

ECSpec1					
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
Reader List	1 Reader	repeatPeriod	M sec	startTrigger	Null
duration	N Sec	stableSetInterval	0 sec	stopTrigger	Null
startTriggerList	Null	stopTriggerList	Null	reportSet	CURRENT
includeCount	No	includeEPC	Yes	includeTag	No
reportIfEmpty	True	reportOnlyOnChange	False	includeSpecInReports	False
includeRawHex	No	includeRawDecimal	No	groupSpec	No
includePatterns	Yes	excludePatterns	No	filterList	No
statProfileNames	No	primaryKeyFields	Null		

ECSpec2 (Same a EPCSpec 1 except for values shown)					
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
includePatterns	No	excludePatterns	Yes	filterList	No

ECSpec3 (Same a EPCSpec 1 except for values shown)					
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
includePatterns	No	excludePatterns	No	filterList*	Yes

*The filterList should contain two filterList members: an include and an exclude member.

- There are no users subscribed to the ECSpec.
- A valid notification URI exists that can accept ECReports from the ALE implementation.

Step	Step description	Expected results
1	Invoke the subscribe method to activate the ECSpec1 A and begin the event cycle.	Subscribe to ECSpec1 A is successful.
2	Put a tag that does not satisfy includefilter condition within time < N sec.	Verify that an empty ECReports is sent to the notification URI after the expiry of N seconds (when the duration timeout occurs).
3	Put a tag that satisfies includefilter condition within time < N sec during the next event cycle.	Verify that immediately an ECReports is sent to the notification URI. The ECReports should conform to a report output per the XSD in the ALE specification and contain the new added tag.

4	Put a tag tag that satisfies include filter condition before the expiration of the repeat period M (Note: N < M).	Verify that no ECReports is sent after the expiry of N seconds.
5	Unsubscribe form ECSpec1 A	Unsubscribe is successful and ECSpec1 A should not longer be active
6	Invoke the subscribe method to activate the ECSpec1 B and begin the event cycle.	Subscribe to ECSpec1B is successful.
7	Put a tag that does not satisfy includefilter condition within time < N sec.	Verify that an empty ECReports is sent to the notification URI after the expiry of N seconds (when the duration timeout occurs).
8	Put a tag that satisfy includefilter condition within time < N sec during the next event cycle.	Verify that an ECReports is sent to the notification URI only after the expiry of N seconds (when the duration timeout occurs). The ECReports should conform to a report output per the XSD in the ALE specification and contain the new added tag.
9	Unsubscribe from ECSpec1 B	Unsubscribe is successful and ECSpec1 B should no longer be active
10	Invoke the subscribe method to activate the ECSpec2 A and begin the event cycle.	Subscribe ECSpec2 A is successful.
11	Put a tag that satisfies excludefilter condition within time < N sec.	Verify that an empty ECReports is sent to the notification URI after the expiry of N seconds (when the duration timeout occurs).
12	Put a tag that does not satisfy excludefilter condition within time < N sec during the next event cycle.	Verify that immediately an ECReports is sent to the notification URI. The ECReports should conform to a report output per the XSD in the ALE specification and contain the new added tag.
13	Put a tag that satisfies excludefilter condition before the expiry of duration, i.e., within M sec of the start of the event cycle in step 3.	Verify that no ECReports is sent after the expiry of N seconds.
14	Unsubscribe from ECSpec2 A	Unsubscribe is successful and ECSpec2 A should no longer be active
15	Invoke the subscribe method to activate the ECSpec2 B and begin the event cycle.	Subscribe ECSpec2 B is successful.
16	Put a tag that satisfy excludefilter condition within time < N sec during the next event cycle.	Verify that an empty ECReports is sent to the notification URI after the expiry of N seconds (when the duration timeout occurs).
17	Put a tag that does not satisfy excludefilter condition within time < N sec.	Verify that an ECReports is sent to the notification URI only after the expiry of N seconds (when the duration timeout occurs). The ECReports should conform to a report output per the XSD in the ALE specification and contain the new added tag.

18	Unsubscribe from ECSpec2 B	Unsubscribe is successful and ECSpec2 B should no longer be active
19	Invoke the subscribe method to activate the ECSpec3 A and begin the event cycle.	Subscribe ECSpec3 A is successful.
20	Put tags that satisfies filter list include and exclude conditions within time < N sec.	Verify that an ECReports only containg the tags that meet the include conditions is sent to the notification immediately (i.e. whenDataAvailble)
21	Put tags that do not satisfy exclude and include conditions within time < N sec during the next event cycle. (Note: Tags from step 20 should be removed before this step is executed.) NOTE: If a tag sarisfies both filter list include and exlude conditions, it must not be reported.	Verify that immediately an ECReports is sent to the notification. The ECReports should conform to a report output per the XSD in the ALE specification and contain the new added tags that did not meet the exclude condition.
22	Unsubscribe from ECSpec3 A	Unsubscribe is successful and ECSpec3 A should no longer be active

530 ## 14.10 TCR-R10 – Using primaryKeyFields, Reading API

531

Using primaryKeyFields, Reading API
TPId: TCR-R10
Requirement Purpose: This Test Case confirms the correct operation of the feature primaryKeyFields of the ECSpec. **Requirements Tested:** GM1, RM1, RM8, RM19, RM21, RM25, RM26, RM64
NOTE: Implementation SHALL support primaryKeyFields list consisting of the single element “epc”. ECSpecValidationException may be thrown for other, unsupported, combinations of primaryKeyFields.

Pre-test conditions:

- A Valid ECSpec (M > N) has been defined as follows:

ECSpec					
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
Reader List	1 Reader	repeatPeriod	M sec	startTrigger, startTriggerList	Null
duration	N Sec	stableSetInterval	0 sec	stopTrigger, stopTriggerList	Null
reportSet	ADDITION	whenDataAvailable	False		
includeCount	No	includeEPC	Yes	includeTag	No
reportIfEmpty	True	reportOnlyOnChange	False	includeSpecInReports	False
includeRawHex	No	includeRawDecimal	No	groupSpec	No
includePatterns	No	excludePatterns	No	primaryKeyFields	@0.32.32
filterList	No	statProfileNames	No		

- There are no users subscribed to the ECSpec.
- The Gen2 tags with unlocked Access Password fields are required.

NOTE: If the ALE Implementation does not support the specified primaryKeyFields value with the specified logical reader, the implemation may raise an ECSpecValidationException in step 2. This response meets conformance requirements. It should be verified the the ECSpecValidationException is not being raised due to another error condition with the ECSpec. However, if the primaryKeyFields list just contains one value, 'epc', an ECSpec validation error should not be raised.

Step	Step description	Expected results
1	Place a Gen2 tag (T1) with a known access password in the reader field.	
2	Invoke the subscribe method to subscribe to the ECSpec providing its HTTP Notification URI	Verify that the an ECReports is received at the end of N sec to the notification URI Verify that ECReport contains information of tag T1. Alternatively, an ECSpecValidationException could be raised. (see note above).
3	Place another Gen2 tag (T2) with the same known access password in the reader field before the start of the next event cycle.	Verify that the an empty ECReports is received at the end of N sec to the notification URI.
4	Place another Gen2 tag (T3) with a different known access password in the reader field before the start of the next event cycle.	Verify that the an ECReports is received at the end of N sec to the notification URI Verify that ECReport contains information of tag T3.
5	Replace the primaryKeyFields of the ECSpec in pre-test condition by a list of {EPC, @0.32.32}. Repeat step 1-4.	Verify that in step 3, an ECReports is received at the end of N sec to the notification URI containing information of tag T2.

6	Repeat step 1-5 with Access Passwords locked for the Gen2 tags T2 and T3.	Verify that all ECReports received are empty except in step 2.
7	Unsubscribe from the current ECSpec.	Verify the unsubscribe is successful
8	Use a new ECSpec that is the same as the one given in the pretest conditions except that primaryKeyFields is changed to just hold one value in its list: 'epc'	
9	Place a Gen2 tag (T1) reader field that is programmed with a valid epc value.	
10	Invoke the subscribe method to subscribe to the ECSpec providing its HTTP Notification URI	Verify that the an ECReports is received at the end of N sec to the notification URI Verify that ECReport contains information of tag T1. Note: it is a conformance requirement failure for a implementation to raise an ECSpecValidation error for this test step (see note above).
11	Unsubscribe from the current ECSpec.	

532

533 14.11 TCR-R11 – Interpretation of *new* in stableSetInterval, 534 Reading API

Interpretation of *new* in stableSetInterval, Reading API
TPId: TCR-R11
Requirement Purpose: This Test confirms that, in context of stablesetinterval, "new" is to be interpreted collectively among readers contributing to this eventcycle is working properly. **Requirements Tested:** GM1, RM1, RM11, RM19, RM21, RM25, RM64

- **Pre-test conditions**: A valid ECSpec has been defined with a stablesetInterval of L seconds and a duration of N second where L< N as shown below.

ECSpec					
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
Reader List	2 Readers	repeatPeriod	0sec	startTrigger, startTriggerList	Null
duration	N	stableSetInterval	L sec	stopTrigger, stopTriggerList	Null
reportSet	CURRENT	primaryKeyFields	Null	whenDataAvailable	False
includeCount	No	includeEPC	Yes	includeTag	No
reportIfEmpty	true	reportOnlyOnChange	false	includeSpecInReports	False
includeRawHex	No	includeRawDecimal	No	groupSpec	No
includePatterns	No	excludePatterns	No		
filterList	No	statProfileNames	No		

- There are no users subscribed to the ECSpec.
- A valid notification URI exists that can accept ECReports from the ALE implementation

Step	Step description	Expected results
1	Put one tag in “Reader1” field.	
2	Invoke the subscribe method to subscribe the user and activate the ECSpec	Subscribe returns void. The user should be subscribed to the ECSpec. The first event cycle should begin.
3	Put the same tag in “reader2” field before L expired.	An ECReports should be sent to the notification URI after L seconds (when the stableSetInterval timeout occurs). The ECReports should conform to a report output per the XSD in the ALE specification and include the added tag.
4	Invoke Unsubscribe	

535

536 14.12 TCR-R12 – Stability of EPC set, Reading API

Stability of EPC set, Reading API
TPId: TCR-R12
Requirement Purpose: This Test confirms that, in the context of stablesetinterval only Additions but not the Deletions are considered in determining that the EPC set is “stable”. **Requirements Tested:** GM1, RM1, RM11, RM26, RM64

Pre-test conditions:

- A valid ECSpec has been defined with a stablesetInterval of L seconds and a duration of N second where L< N as shown below.

ECSpec					
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
Reader List	1 Reader	repeatPeriod	0sec	startTrigger, startTriggerList	Null
duration	N	stableSetInterval	L sec	stopTrigger, stopTriggerList	Null
reportSet	CURRENT	primaryKeyFields	Null	whenDataAvailable	False
includeCount	No	includeEPC	Yes	includeTag	No
reportIfEmpty	True	reportOnlyOnChange	False	includeSpecInReports	False
includeRawHex	No	includeRawDecimal	No	groupSpec	No
includePatterns	No	excludePatterns	No		
filterList	No	statProfileNames	No		

- There are no users subscribed to the ECSpec.
- A valid notification URI exists that can accept ECReports from the ALE implementation

Step	Step description	Expected results
1	Invoke the subscribe method to subscribe the user and activate the ECSpec	Subscribe returns void. The user should be subscribed to the ECSpec. The first event cycle should begin.
2	Continually add tags to the reader field at a rate faster than one tag per L seconds for a period longer than N seconds.	An ECReports should be sent to the notification URI after N seconds (when the duration timeout occurs). The ECReports should conform to a report output per the XSD in the ALE specification and include all added tags.
3	During the next eventcycle remove tags from the reader field at a rate faster than one tag per L seconds.	An ECReports should be sent to the notification URI after L seconds (when the stableset timeout occurs). The ECReports should conform to a report output per the XSD in the ALE specification. And include all tags.
4	Invoke Unsubscribe	

537

538 14.13 TCR-R13 – includeSpecInReports, Reading API

includeSpecInReports, Reading API
TPId: TCR-R13
Requirement Purpose: This Test confirms that if includeSpecInReports is true in ECSpec definition then every ECReport instance must include the complete ECSpec as a part of ECReport. **Requirements Tested:** GM1, RM1, RM6, RM23, RM40

Pre-test conditions:

- A valid ECSpec has been defined with a repeatPeriod of M seconds and a duration of N second where M > N as shown below.

ECSpec					
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
Reader List	1 Reader	repeatPeriod	M sec	startTrigger, startTriggerList	omitted
duration	N sec	stableSetInterval	0	stopTrigger, stopTriggerList	omitted
reportSet	CURRENT	primaryKeyFields	Null	whenDataAvailable	False
includeCount	Yes	includeEPC	Yes	includeTag	No
reportIfEmpty	True	reportOnlyOnChange	True	includeSpecInreports	True
includeRawHex	No	includeRawDecimal	No	groupSpec	No
includePatterns	No	excludePatterns	No		
filterList	No	statProfileNames	No		

Step	Step description	Expected results
1	Define an ECSpec according to the conditions specified in pre-test.	One ECSpec object of the specified name will be defined in ALE middleware.
2	Place a set of tag in the reader field.	
3	Invoke poll using the name of defined ECSpec	After N seconds, an ECReport that conforms the ale XSD should be returned listing all tags in the reader field. And the report should include the ECSpec also.

539

540 14.14 TCR-R14 – stableSetInterval and duration, Reading API

stableSetInterval and duration, Reading API
TPId: TCR-R14
Requirement Purpose: This Test Case confirms the correct operation of the stableSetInterval and repeatPeriod features of the ECSpec. It also tests the includeTag for the ECReport and the reportIfEmpty field. **Requirements Tested:** GM1, RM1, RM11, RM64

Pre-test conditions:

- A valid ECSpec has been defined with a repeatPeriod of M seconds and a stableSetInterval of N second where M > N as shown below.

ECSpec					
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
Reader List	1 Reader	repeatPeriod	M sec	startTrigger, startTriggerList	Null
duration	0	stableSetInterval	N sec	stopTrigger, stopTriggerList	Null
reportSet	CURRENT	primaryKeyFields	Null	whenDataAvailable	False
includeCount	No	includeEPC	Yes	includeTag	No
reportIfEmpty	True	reportOnlyOnChange	False	includeSpecInreports	False
includeRawHex	No	includeRawDecimal	No	groupSpec	No
includePatterns	No	excludePatterns	No		
filterList	No	statProfileNames	No		

- There are no users subscribed to the ECSpec.
- A valid notification URI exists that can accept ECReports from the ALE implementation

Step	Step description	Expected results
1	Invoke the subscribe method to activate the ECSpec and begin the event cycle	Subscribe returns void
2	Continually add tags to the reader field at a rate faster than one tag per N seconds for a period longer than M seconds.	No ECReports should be returned at a time less than M nor at a time greater than M
3	Stop the introduction of new tags.	N seconds after the introduction of new tags is stopped, confirm an ECReports that conforms to the ALE XSD was returned and contained Tag EPCs for all tags introduced during step 2.

541

542 14.15 TCR-R15 – Additions, RepeatPeriod and duration, Reading 543 API

Additions, RepeatPeriod and duration, Reading API
TPId: TCR-R15
Requirement Purpose: This Test Case Requirement confirms the tag Additions, duration, repeatPeriod, includeRawHex and reportOnlyOnChange=true features of the ECSpec. The test will verify the correct operation of the repeatPeriod and duration features for the cases where repeatPeriod > duration and repeatPeriod < duration. **Requirements Tested:** GM1, RM1, RM11, RM22, RM26, RM39, RM41, RM43, RM64

Pre-test conditions:

- A valid ECSpec has been defined with a repeatPeriod of M seconds and a duration of N second where M > N as shown below.

ECSpec					
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
Reader List	1 Reader	repeatPeriod	M sec	startTrigger, startTriggerList	omitted
duration	N sec	stableSetInterval	0	stopTrigger, stopTriggerList	omitted
reportSet	ADDITIONS	primaryKeyFields	Null	whenDataAvailable	False
includeCount	No	includeEPC	No	includeTag	No
reportIfEmpty	True	reportOnlyOnChange	True	includeSpecInreports	False
includeRawHex	Yes	includeRawDecimal	No	groupSpec	No
includePatterns	No	excludePatterns	No		
filterList	No	statProfileNames	No		

- There are no users subscribed to the ECSpec.
- A valid notification URI exists that can accept ECReports from the ALE implementation.

Step	Step description	Expected results
1	Invoke the subscribe method to subscribe the user and activate the ECSpec	Subscribe returns void. The user should be subscribed to the ECSpec. The first event cycle should begin.
2	Move a set of tags into the reader field	An ECReports should be sent to the notification URI after N seconds (when the duration timeout occurs). The ECReports should conform to a report output per the XSD in the ALE specification and all tag EPCs should be represented in raw hex format.
3	Remove some but not all of the tags in the set starting before the next event cycle begins.	A new event cycle should begin after the repeatPeriod of M second is reached, measured from the *start* of the previous event cycle. Another ECReports that conforms to the ALE XSD should be sent to the notification URI after N seconds after the start of the event cycle. The ECReports should conform to a report output per the ALE specification. The report should include an ECReport instance but that ECReport should be empty since no tags were added.

4	Add the tags that were removed in step 3.	A new event cycle should begin after the repeatPeriod of M second is reached, measured from the start of the previous event cycle. Another ECReports should be sent to the notification URI after N seconds after the start of the event cycle. The ECReports should conform to the ALE XSD. The report should contain the tag EPCs that were added back.
5	Keep the tags in the reader field the same. Do not add or remove any tags.	A new event cycle should begin after the repeatPeriod of M second is reached, measured from the start of the previous event cycle. No ECReports should be received by the notification URI either before or after the duration period expires or before or after the repeatPeriod expires.
6	Unsubscribe from the current ECSpec	Unsubscribe returns void. The user should be unsubscribed the ECSpec should be inactive.
7	Invoke the subscribe method using a new ECSpec that sets the duration period N is greater than the repeatPeriod M. All other ECSpec parameter should be the same as before.	Subscribe returns void. The user should be subscribed to the new ECSpec. The first event cycle should begin.
8	Move a set of tags into the reader field	An ECReports that conforms to the ALE XSD should be sent to the notification URI after N seconds (when the duration timeout occurs). The ECReports should conform to a report output per the ALE specification and all tag EPCs should be represented in raw hex format. The next event cycle should begin immediately.
9	Add tags to the reader field	Another ECReports that conforms to the ALE XSD should be sent to the notification URI after N seconds (when the duration timeout occurs). The ECReports should conform to a report output per the ALE specification and contain just the EPCs that were added represented in raw hex format. The next event cycle should begin immediately.
10	Unsubscribe from the ECSpec	Unsubscribe returns void. The user should be unsubscribed the ECSpec should be inactive.

544

545 14.16 TCR-R16 – Include Filter, Groups and Multiple Readers, 546 Reading API

Include Filter, Groups and Multiple Readers, Reading API
TPId: TCR-R16

Requirement Purpose: This Test Case confirms the include filter, groupSpecs, multiple logical reader support, reportOnlyOnChange, includeCount and multiple ECReport instances in an ECReports.
Requirements Tested: GM1, GM92, RM1, RM27, RM29, RM30, RM32, RM34, RM39, RM40, RM41, RM43, RM54, RM55

Pre-test conditions:

- A valid ECSpec has been defined with an include filter, groupSpec, multiple logical readers in the reader list, and multiple ECReportSpec(s) in the reportSet list. The repeatPeriod M equals the duration N (M=N).
- There are no users subscribed to the ECSpec.
- A tag set suitable to properly exercise the groupSpec and Filter Pattern features is ready. For tests that verify the groupSpec feature, the tag set should be such that there are tags that will match the patterns so they will fall into specified groups and there will also be tags that do not match any patterns and thus be placed in the default group.
  A valid notification URI exists that can accept ECReports from the ALE implementation.

ECSpec					
includeSpecInReports = Yes					
Boundary Parameters					
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
Reader List	2 Readers	repeatPeriod	M sec	startTrigger, startTriggerList	Null
duration	N sec	stableSetInterval	0	stopTrigger, stopTriggerList	Null
ECReport Instance 1					
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
reportSet	CURRENT	primaryKeyFields	Null	whenDataAvailable	False
includeCount	Yes	includeEPC	Yes	includeTag	No
reportIfEmpty	False	reportOnlyOnChange	True		
includeRawHex	No	includeRawDecimal	No	groupSpec	Yes
includePatterns	No	excludePatterns	No	fieldSpec	epc, EPC, epc-tag
filterList	No	statProfileNames	No		
ECReport Instance 2					
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
reportSet	CURRENT	primaryKeyFields	Null	whenDataAvailable	False
includeCount	No	includeEPC	No	includeTag	Yes
reportIfEmpty	True	reportOnlyOnChange	False		
includeRawHex	No	includeRawDecimal	No	groupSpec	No
includePatterns	Yes	excludePatterns	No		
filterList	No	statProfileNames	No		

Step	Step description	Expected results
1	Invoke the subscribe method to subscribe the user and activate the ECSpec	Subscribe returns void. The user should be subscribed to the ECSpec. The first event cycle should begin.
2	The set of tags should be in the reader field of at least one of the logical readers.	The event cycle should end after the duration period time expires. ECReports that conforms to the ALE XSD should be sent to the notification URI. The next event cycle should be begin immediately after the duration period expires. An ECReports should be sent to the notification URI. The ECReports should conform to a report output per the ALE specification. There should be two ECReport instances in the ECReports. The first instance should contain all the tags grouped according the groupSpec with a count provided for each group. The second report instance should contain a list of Tag EPCs that have passed the include filter.
3	Remove all of the tags that would pass the include filter. Only tags that would be filtered out should remain in the tag set.	ECReports that conforms to the ALE XSD should be sent to the notification URI. The event cycle should end after the duration period time expires. The next event cycle should be begin immediately after the duration period expires. An ECReports should be sent to the notification URI. The ECReports should conform to a report output per the ALE specification. There should be two ECReport instances in the ECReports. The first instance should contain all the tags in the grouped according the groupSpec with a count provided for each group. There second report instance should be present but empty.
4	The tag set should remain unchanged from step 3 for the next event cycle.	The event cycle should end after the duration period time expires. ECReports that conforms to the ALE XSD should be sent to the notification URI. The next event cycle should be begin immediately after the duration period expires. An ECReports should be sent to the notification URI. The ECReports should conform to a report output per the ALE specification. There should be only one ECReport instance in the ECReports for the ECReport instance 2. It should be empty.

5	Add tags back to the tag set for the next event cycle. The tags should be able to pass the include filter.	The event cycle should end after the duration period time expires. ECReports that conforms to the ALE XSD should be sent to the notification URI. The next event cycle should be begin immediately after the duration period expires. An ECReports should be sent to the notification URI. The ECReports should conform to a report output per the ALE specification. There should be two ECReport instances in the ECReports. The first instance should contain all the tags grouped according the groupSpec with a count provided for each group. The second report instance should contain a list of Tag EPCs that have passed the include filter.
6	Unsubscribe from the ECSpec	Unsubscribe returns void. The user should be unsubscribed the ECSpec should be inactive.

547

548 ## 14.17 TCR-R17 – Include Filtering, Reading API

549

Include Filtering, Reading API
TPId: TCR-R17
Requirement Purpose: This Test Case Requirement confirms the following features of the ECSpec: include count, include current, includeTag format, includeRawDecimal format, reportIfEmpty=false, include pattern filtering and one logical reader in the reader list. **Requirements Tested:** GM1, GM26, GM27, GM28, GM29, GM30, GM33, GM34, GM35, GM36, GM37, GM38, GM39, GM40, GM42, GM43, GM44, GM46, GM47, GM48, GM50, GM61, GM63, GM65, GM66, GM67, GM68, GM69, GM71, GM82, GM84, GM85, GM86, GM87, GM88, GM89, GM90, GM91, GM92, GM93, GM94, GM96, GM97, GM98, GM99, GM100, GM101, GM102, GM103, GM104, GM105, GM106, GM107, GM108, GM109, GM110, GM111, GM112, RM1, RM9, RM27, RM29, RM30, RM34, RM40, RM45

Pre-test conditions:

- A valid ECSpec has been defined as shown. The repeatPeriod = the duration (M = N)

ECSpec					
Parameter	Value	Parameter	Value	Parameter	Value
Reader List	1 Reader	repeatPeriod	M	startTrigger, startTriggerList	omitted
duration	N	stableSetInterval	0	stopTrigger, stopTriggerList	omitted
reportSet	CURRENT	primaryKeyFields	Null	whenDataAvailable	False
includeCount	Yes	includeEPC	No	includeTag	Yes
reportIfEmpty	False	reportOnlyOnChange	False	includeSpecInReports	False
includeRawHex	No	includeRawDecimal	Yes	groupSpec	No
includePatterns	Yes	excludePatterns	No		
filterList	Yes	statProfileNames	No		

ECFilterListMember					
Parameter	Value	Parameter	Value	Parameter	Value
includeExclude	Include	Fieldspec	ECFieldspec1/ ECFieldspec2/ ECFieldspec3/ ECFieldspec4	patList	*valid pattern list*

ECFieldSpec1					
Parameter	Value	Parameter	Value	Parameter	Value
Fieldname	@1.96.32	Datatype	uint	Format	hex

ECFieldSpec2					
Parameter	Value	Parameter	Value	Parameter	Value
Fieldname	@4.1.0	Datatype	uint	Format	hex

ECFieldSpec3					
Parameter	Value	Parameter	Value	Parameter	Value
Fieldname	epc	Datatype	epc	Format	epc-tag

ECFieldSpec4					
Parameter	Value	Parameter	Value	Parameter	Value
Fieldname	killPwd	Datatype	omitted	Format	omitted

ECFieldSpec5					
Parameter	Value	Parameter	Value	Parameter	Value
Fieldname	accessPwd	Datatype	omitted	Format	omitted

ECFieldSpec6					
Parameter	Value	Parameter	Value	Parameter	Value
Fieldname	userDefined*	Datatype	omitted	Format	omitted

*****userDefined refers to a filed name defined using the TM API

- There are no users subscribed to the ECSpec.
- A tag set of Gen 2 tags that will have some but not all tags pass the include filter is required.

Step	Step description	Expected results
1	Invoke the poll method to activate the ECSpec (using **ECFieldSpec1**) and begin the event cycle. Ensure that the tag set is in the reader field.	The event cycle should have started.
2	Verify that after time M, when the repeatPeriod expires, an ECReports is returned by the poll.	An ECReports that conforms to the ALE XSD should be returned by the poll in Step 1. It should include those tags from the tag set that matched the include filter. The tag identities should be provided in Tag and Raw Decimal Format and should be consistent with the EPCs on the tags. A count of the tags should be in the report.
3	Invoke the poll method again but only have tags in the tag set presented to the reader field where all tags that will be filtered out by include filter.	After the repeatPeriod expires after time M, an ECReports that conforms to the ALE XSD should be returned. It should include an empty ECReports.
4	Repeat steps 1-3 with **ECFieldSpec2**	Verify that the ECReport omits value field in ECReportMemberField because of "field not found" condition.
5	Repeat steps 1-3 with **ECFieldSpec3**	Verify that the tags with the same epc value are included in the ECReport.
6	Repeat steps 1-3 with **ECFieldSpec4**	Verify that the tags with the same killpwd value are included in the ECReport.

7	Repeat steps 1-3 with **ECFieldSpec5**	Verify that the tags with the same accesspwd value are included in the ECReport.
8	Repeat steps 1-3 with ECFieldSpec6 (optional)	Verify that the tags with the userDefined value are included in the ECReport. Note: the tags could be excluded as a result of raising the "operation not possible" condition because the ALE implementation does not support tidBank.

550

551 14.18 TCR-R18 – Reporting Variable Fields, Reading API

Reporting Variable Fields, Reading API
TPId: TCR-R18
Requirement Purpose: This Test Case demonstrates the variable fieldname feature which including the ECReportOutputFieldSpec report, and the testing of the variable fieldname syntax. The test result will be different for those implementations that fully support the feature and those that only recognize the syntax. **Requirements Tested:** GM1, GM72, GM73, GM74, GM75, GM78, GM79, GM80, GM81, RM1, RM23, RM44

Pre-test conditions:

- A tag with a user memory bank that has been correctly encoded according to ISO 15962.
- ECSpec and ECReportOutputFieldSpecs for the test:

ECSpec					
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
Reader List	1 Reader	repeatPeriod	M sec	startTrigger, sartTriggerList	Null
duration	N sec	stableSetInterval	0	stopTrigger, stopTriggerlIst	Null
reportSet	CURRENT	primaryKeyFields	Null	whenDataAvailable	False
includeCount	No	includeEPC	No	includeTag	No
reportIfEmpty	True	reportOnlyOnChange	False	includeSpecInReports	False
includeRawHex	No	includeRawDecimal	No	groupSpec	No
includePatterns	No	excludePatterns	No		
filterList	No	statProfileNames	No		

ECReportOutputFieldSpec A	
Parameter	**Value**
FieldSpec	@3.urn:oid:1.0.15961.12.4
Name	IssuingStation
Include FieldSpec	No
Datatype	Omitted
Format	Omitted

ECReportOutputFieldSpec B	
Parameter	**Value**
FieldSpec	@3.urn:oid:1.0.15961.12.*
Name	omitted
Include FieldSpec	No
Datatype	Omitted
Format	Omitted

ECReportOutputFieldSpec C	
Parameter	**Value**
FieldSpec	@0.urn:oid:1.0.15961.12.*
Name	omitted
Include FieldSpec	No
Datatype	Omitted
Format	Omitted

	ECReportOutputFieldSpec D	
	Parameter	**Value**
	FieldSpec	@2.urn:oid:1.0.15961.12.*
	Name	omitted
	Include FieldSpec	No
	Datatype	Omitted
	Format	Omitted

Step	Step description	Expected results
1	Place a tag in the reader field	
2	Invoke the immediate method using the ECSpec with ECReportOutputFieldSpecA as specified in the pre-test conditions.	An ECReport should be returned that includes the variable field specified. The field is not returned if the implementation does not support variable fields.
3	Invoke the immediate method using the ECSpec with ECReportOutputFieldSpecB as specified in the pre-test conditions.	An ECReport should be returned that includes all variable fields encoded in the tag's user memory bank. The field is not returned if the implementation does not support variable fields
4	Invoke the immediate method using the ECSpec with ECReportOutputFieldSpecC as specified in the pre-test conditions.	An ECReport should be returned that raises the "field not found" condition. It could also raise the "operation not found" condition if the implementation does not support variable fields. In either case the condition will result in omitting the relavent field.
5	Invoke the immediate method using the ECSpec with ECReportOutputFieldSpecD as specified in the pre-test conditions.	An ECReport should be returned that raises the "field not found" condition. It could also raise the "operation not found" condition if the implementation does not support variable fields. In either case the condition will result in omitting the relavent field.

552 ## 14.19 TCR-R19 – Initiation and Termination Conditions for
553 ## Undefining an ECSpec during Active Poll, Reading API

554

Initiation and Termination Conditions for Undefining an Active Poll, Reading API		
TPId: TCR-R19		
Requirement Purpose: This Test Case Requirement verifies that the initiation and termination conditions are properly filled when an active ECSpec, as a result of a Poll call, is undefined. **Requirements Tested:** GM1, GM17, RM1, RM48		
Pre-test conditions: • A valid ECSpec has been defined with repeatPeriod = 0 sec, duration = 30 sec.		
Step	**Step description**	**Expected results**

1	Invoke the poll method using the ECSpec defined in pre-test condition.	Poll call is outstanding.
2	Wait for 10 sec.	-
3	Invoke ndefined method to ndefined the ECSpec defined in step 1.	Verify that an ECReports is received right after the ndefined call with InitiationCondition = REQUESTED and TerminationCondition = UNDEFINE.

555

556 ## 14.20 TCR-R20 – Realtime Clock Trigger

557

Realtime Clock Trigger		
TPId: TCR-R20		
Requirement Purpose: This Test Case Requirement verifies the proper operation of the realtime clock trigger.. **Requirements Tested:** GM1, RM1, RM17, RM18		
Pre-test conditions: • A valid ECSpec has been defined with duration = N seconds and a real-time clock trigger for the start trigger. The period for the real-time clock trigger should be greater than N seconds with an offset equal 0 and a timezone set to the local timezone. Report if empty should be true.		
Step	**Step description**	**Expected results**
1	Invoke the subscribe method using the ECSpec defined in pre-test condition using a valid using a valid notification URI.	A subscription should be active
2	Wait for number of milliseconds past midnight modulo period equals offset plus N seconds	Verify an ECReports is sent to the notification URI N seconds after the real-time clock start trigger was set off.
3	Invoke ndefined method to ndefined the ECSpec defined in step 1.	

558

559 ## 14.21 TCR-R21 – XML Vendor Extension Validaion

XML Vendor Extension Validation
TPId: TCR-R21
Requirement Purpose: This Test Case confirms that vendor extensions to the ECSpec and ECReports have been added in accordance with the rules set forth in the ALE 1.1 specification. This TCR is opational. This TCR only needs to be executed for implementation that have vendor extension. **Requirements Tested:** XM1, XM2, XM3, XM4, XM5, XM6, XM8, XM9, XM12

Pre-test conditions: • The vendor has submitted XML files containing instances of ECSpecs and ECReports that contain the the vendor extensions or the vendor's XSD for the Reading API or appropriate documentation confirming the vendor is the owner of the namespace used for the vendor extensions. • The vendor has provided its XSD files so they can be inspected to ensure that elements, attributes and extensions have not be added in places not allowed by the specification.		
Step	**Step description**	**Expected results**
1	Examine the XML documents, XSD documents or other documentation submitted by the vendor to verify the vendor is the owning authority for the name space used for all vendor attribute and element extensions.	Confirm (by design)
2	Validate the XML ECReports and ECSpec instance documents received in TCR-R1 through R20 against the ALE 1.1 Reading API XSD. (See section 13.4)	The XML documents should validate successfully.
3	Inspect the vendor XSDs to ensure that elements, attributes and extensions have not be added in places not allowed by the ALE 1.1 specification.	There are no elements, attributes and extensions added in the vendor's XSDs in places not allowed by the ALE 1.1 specificatoin.

560

561

15 Writing API

15.1 TCR-W1 – Get Version, Writing API

Get Version, Writing API		
TPId: TCR-W1		
Requirement Purpose: This Test Case confirms the proper functions of the ALE methods of the Writing API that return the ALE standard version and the vendor version for the ALE implementation under test. The return of correct version numbers also confirms the correct implementation is being tested. **Requirements Tested:** GM1, GM2, GM3, GM4, GM5, WM1		
Pre-test conditions: • None		
Step	**Step description**	**Expected results**
1	Invoke the getStandardVersion method of the Writing API	Confirm the string "1.1" is returned.
2	Invoke the getVendorVersion method of the Writing API	• Confirm that either an empty string or a string conforming to a proper URI is returned. • Confirm the vendor is the owning authority of the URI if the returned string is not empty (by Design) • Confirm the result returned by this method only pertain to the API to the Writing API.

15.2 TCR-W2 – Defining, Un-defining, Retrieving CCSpecs, Writing API

Defining, Un-defining, Retrieving CCSpecs, Writing API		
TPId: TCR-W2		
Requirement Purpose: This Test Case confirms that a valid CCSpec can be defined and undefined. Further the defining and un-defining of the CCSpec can be verified with ALE API methods getCCSpec and getCCSpecNames. **Requirements Tested**: GM1, GM6, GM7, GM12, WM1, WM2, WM3		
Pre-test conditions: • No CCSpecs are defined. • Ensure all specName parameters accept as a name any non-empty string of Unicode characters that does not include Pattern_White_Space or Pattern_Syntax characters (see GM6) • For step 7, the ALE implementation should support reading APIs.		
Step	**Step description**	**Expected results**
1	Invoke the define method with a valid CCSpec without extensions	The ALE implementation contains the CCSpec definition supplied in the define method. Steps 2 and 3 confirm the defining of the CCSpec.

2	Verify the CCSpec was defined by invoking the getCCSpecNames method	Verify that the name returned in the list is that of the CCSpec defined in step 1.
3	Invoke getCCSpec using the name of the defined CCSpec.	Verify that the CCSpec returned is equivalent to the one defined
4	Invoke ndefined to remove the CCSpec that was defined.	The ALE implementation should no longer have the CCSpec defined. Confirmed by step 5.
5	Verify that the CCSpec is undefined by invoking the getCCSpecNames method.	Verify that the list returned is empty.
6	Repeat steps 1 through 5 for a valid CCSpec with extensions.	Note that when Step 3 is repeated, the CCSpec returned by getCCSpec may not necessarily include any of the extension elements provided in Step 1, if those extensions are not understood by the implementation.
7	Invoke the define method with an ECSpec and specName = "foo". Invoke the define method with a CCSpec and specName = "foo". (optional – only if the implementation implanted both the reading and writing APIs.)	Verify that the ALE implementation does accept both the ECSpec and the CCSpec and does not raise a DuplicateNameException.

569

570 15.3 TCR-W3 – Exceptions, Writing API

571

Exceptions, Writing API		
TPId: TCR-W3		
Requirement Purpose: This Test Case confirms that the ALE implementation will raise all exceptions as defined in the ALE specification. This covers exceptions raised due to incorrect parameters passed in ALE API methods and exceptions raised due to missing or invalid parameters in an CCSpec. **Requirements Tested:** GM1, GM6, GM8, GM32, GM38, GM42, GM46, GM50, GM55, GM60, GM64, GM83, RM31, WM1, WM5, WM8, WM10, WM13, WM18, WM22, WM29, WM40, WM43, WM57, WM63, WM65, WM70		
Pre-test conditions: • No CCSpecs are defined Note: The CCSpecs used in this Test Case Requirement should be valid except for the conditions specified in step being performed.		
Step	**Step description**	**Expected results**
1	Invoke the getCCSpec with an unknown spec name.	Verify that the ALE implementation raises a NoSuchNameException.
2	Invoke the poll method using an unknown name for the CCSpec string.	Verify that the ALE implementation raises a NoSuchNameException.
3	Invoke the subscribe method with an unknown CCSpec name	Verify that the ALE implementation raises a NoSuchNameException.
4	Invoke the unsubscribe method with a defined CCSpec name and a well formed notification URI. The notification URI should not belong to a user who is subscribed	Verify that the ALE implementation raises a NoSuchSubscriberException.

5	Invoke the subscribe method using the name of a valid and defined CCSpec and a well formed notification URI that is not supported by the implementation under test	Verify that the ALE Implementation raises an InvalidURIException
6	Invoke the unsubscribe method with an unknown CCSpec name	Verify that the ALE implementation raises a NoSuchNameException.
7	Invoke the getSubscribers method with an unknown CCSpec name	Verify that the ALE implementation raises a NoSuchNameException.
8	Invoke the ndefined method with an unknown CCSpec name.	Verify that the ALE implementation raises a NoSuchNameException.
9	Invoke the subscribe method with an non-conforming URI	Verify that the ALE Implementation raises an InvalidURIException
10	Invoke the unsubscribe method with a defined CCSpec name and an non-conforming URI	Verify that the ALE Implementation raises an InvalidURIException
11	Invoke the define method with a valid CCSpec	The ALE implementation holds the CCSpec definition
12	Verify the CCSpec was defined by invoking a getCCSpecNames method	Verify that the name returned is that of the CCSpec just defined
13	Invoke the define method again with a valid CCSpec and the name of the CCSpec defined in step 11.	Verify that the ALE implementation raises a DuplicateNameException
14	The same subscriber should subscribe to the CCSpec to which the subscriber is already subscribed.	Verify that the ALE implementation raises a DuplicateSubscriptionException
15	Invoke the immediate method with a CCSpec that has a logicalReaders list which is either null, or omitted, or an empty list, or contains names that are unknown to the ALE implementation.	Verify that the ALE implementation raises an CCSpecValidationException
16	Invoke the define method with a CCSpec that has a CCBoundarySpec parameter that is null or omitted.	Verify that the ALE implementation raises a CCSpecValidationException
17	Invoke the define method with a CCSpec that has a tagsProcessedCount of CCBoundarySpec parameter that is negative.	Verify that the ALE implementation raises a CCSpecValidationException
18	Invoke the define method with a CCSpec whose CCBoundarySpec contains a duration, repeatPeriod, or noNewTagsInterval parameter that is negative	Verify that the ALE implementation raises a CCSpecValidationException
19	Invoke the define method with a CCSpec whose CCBoundarySpec has no termination condition except afterError.	Verify that the ALE implementation raises a CCSpecValidationException
20	Invoke the define method with two CCCmdSpec instances with identical name fields.	Verify that the ALE implementation raises a CCSpecValidationException

21	Invoke the immediate method with a CCSpec whose CCBoundarySpec has a startTrigger or a stopTrigger which has a value that does not conform to the URI syntax.	Verify that the ALE implementation raises a CCSpecValidationException
22	Invoke the define method with a CCSpec which has the patList parameter of ECFilterListMember instance empty, null or omitted.	Verify that the ALE implementation raises a CCSpecValidationException
23	Invoke the define method with a CCSpec whose CCOpSpec has a opType parameter which is not a standard opType value.	Verify that the ALE implementation raises a CCSpecValidationException
24	Invoke the define method with a CCSpec whose CCOpSpec has a opType parameter which requires a fieldspec, and fieldspec is null or omitted.	Verify that the ALE implementation raises a CCSpecValidationException
25	Invoke the define method with a CCSpec whose CCOpSpec has an opType parameter which does not require a fieldspec, and fieldspec is specified.	Verify that the ALE implementation raises a CCSpecValidationException.
26.	Invoke the define method with a CCSpec whose CCOpSpec has an opType parameter which requires a dataSpec, and dataSpec is null or omitted.	Verify that the ALE implementation raises a CCSpecValidationException.
27	Invoke the define method with a CCSpec whose CCOpSpec has an opType parameter which does not require a dataSpec, and dataSpec is specified.	Verify that the ALE implementation raises a CCSpecValidationException.
28	Invoke the define method with a CCSpec whose CCOpSpec has an opType parameter whose dataSpec parameter specifies a value that is invalid for the specified operation.	Verify that the ALE implementation raises a CCSpecValidationException.
29	Invoke the define method with a CCSpec whose statProfileNames contain an element with unknown statistics profile	Verify that the ALE implementation raises a CCSpecValidationException
30	Invoke the define method with a CCSpec whose CCFilterSpec has filterList with patList that does not conform to syntax rules for patterns	Verify that the ALE implementation raises a CCSpecValidationException
31	Invoke the define method with a CCSpec whose CCFilterSpec has filterList with fieldspec with unknown datatype and format	Verify that the ALE implementation raises a CCSpecValidationException

32	Invoke the define method with a specName that uses a diacritical letter (e.g. embarcadère). Then invoke the ndefined method with a specName that looks equivalent but does not contain the diacritical mark (e.g. embarcadere)	Verify that the ALE implementation raises a NoSuchNameException
33	Invoke the poll method which has two or more CCParameterEntryList intstances wit the same name.	Verify that the ParameterException is raised.
34	Invoke the immediate method which includes a CCOpDataSpec of type PARAMETER.	Verify that the ParameterForbidden Exception is raised.
35	Define a CCSpec that includes a CCOpDataSpec of type PARAMETER and then Invoke the the subscribe method using the CCSpec just defined.	Verify that the ParameterForbidden Exception is raised.

572
573

574 15.4 TCR-W4 – Subscribe and Unsubscribe for READ Operation, 575 Writing API

576

Subscribe and Unsubscribe for READ Operation, Writing API					
TPId: TCR-W4					
Requirement Purpose: This Test Case confirms that clients can subscribe and unsubscribe to CCSpecs that have been correctly defined and the notification URIs used conforms to the ALE1.1 standard. Multiple subscriptions to the same CCSpec are tested. **Requirements Tested:** GM1, GM9, GM13, GM14, GM15, GM16, GM18, GM84, WM1, WM2, WM6, WM9, WM11, WM15, WM38, WM44, WM45, WM46, WM47, WM48, WM49, WM50, WM51, WM52, WM75, XM15, XM16, XM17, XM18					
Pre-test conditions: • A valid CCSpec has been defined as follows:					
CCSpec					
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
Reader List	1 Reader*	includeSpecInReports	False	startTriggerList	Null
duration	N Sec	repeatPeriod	M sec	stopTriggerList	Null
noNewTagsInterval	0	tagsProcessedCount	0	afterError	False
reportIfEmpty	False	statProfileNames	Null	filterList	No
opType	READ	fieldspec	epc	dataspec	Null
• The reader name string should be equal to the CCSpec name string (Test GM9) • A tag set is placed in the reader field					
Step	**Step description**	**Expected results**			

1	The first subscriber invokes the subscribe method to subscribe to the CCSpec providing its HTTP Notification URI	A user is subscribed to the CCSpec in the subscribe invocation and the associated command cycle is activated. Step 2 provides verification.
2	Invoke the getSubscribers method to verify that the subscribe method succeeded.	getSubscriber returns the list of notification URIs. The notification URI from step 1 should be in the list. Verify that the correct CCReports are being received at the notification URI per the boundary condition specified in the CCSpec
3	A second subscriber invokes the subscribe method to subscribe to the same CCSpec as step 1 therefore providing its HTTP Notification URI.	A second user is subscribed to the CCSpec in the subscribe invocation. Step 4 provides verification.
4	Invoke the getSubscriber method to verify that the subscribe method succeeded.	getSubscriber returns the list of notification URIs. The notification URI from step 1 and step 3 should be in the list. Verify that the correct CCReports are being received at the notification URI per the boundary condition specified in the CCSpec
5	The first subscriber un-subscribes by invoking the unsubscribe method using its HTTP Notification URI as a parameter.	The first user is unsubscribed.
6	Invoke the getSubscribers method to verify that the unsubscribe method succeeded and the first subscriber is no longer subscribed.	getSubscriber returns the list of notification URIs. The notification URI from step 1 should no longer be in the list.
7	The second subscriber un-subscribes by invoking the unsubscribe method.	The second user is unsubscribed
8	Invoke the getSubscribers method to verify that the unsubscribe method succeeded and the second subscriber is no longer subscribed.	getSubscriber returns the list of notification URIs. The notification URI from step 3 should no longer be in the list.
9	Repeat steps 1 through 8 replacing the HTTP Notification URIs with a TCP Notification URIs	
10	Repeat steps 1 through 8 replacing the HTTP Notification URIs with a File Notification URIs	

577

578 15.5 TCR-W5 – Subscribe and Unsubscribe for WRITE and LOCK
579 operations, Writing API

580

Subscribe and Unsubscribe for WRITE and LOCK operations, Writing API
TPId: TCR-W5

Requirement Purpose: This Test Case confirms that clients can subscribe and unsubscribe to CCSpecs that have been correctly defined and the notification URIs used conform to the ALE1.1 standard. Functionality of the field tagsProcessedCount is also verified.
Requirements Tested: GM1, GM13, GM14, GM15, GM16, GM18, GM28, GM31, GM85, GM86, GM87, GM88, GM89, GM90, GM91, GM93, GM94, GM96, GM97, GM98, GM99, GM100, GM101, GM102, GM103, GM104, GM105, GM106, GM107, GM108, GM109, GM110, GM111, GM112, WM1, WM2, WM6, WM9, WM11, WM14, WM15, WM17, WM19, WM21, WM31, WM32, WM34, WM35, WM36, WM38, WM39, WM41, WM42, WM44, WM45, WM46, WM47, WM48, WM49, WM50, WM51, WM52, WM58, WM75

Pre-test conditions:

- Three valid CCSpecs have been defined as follows:

CCSpec1					
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
Reader List	1 Reader	includeSpecInReports	False	startTriggerList	Null
duration	0	repeatPeriod	0	stopTriggerList	Null
noNewTagsInterval	0	tagsProcessedCount	1	afterError	False
reportIfEmpty	False	statProfileNames	Null	filterList	No
opType	WRITE	fieldspec	epcBank	opDataSpecType	LITERAL
data	*hex value for epcBank*	fieldSpec data type and format	default		

CCSpec2					
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
Reader List	1 Reader	includeSpecInReports	False	startTriggerList	Null
duration	0	repeatPeriod	0	stopTriggerList	Null
noNewTagsInterval	0	tagsProcessedCount	1	afterError	False
reportIfEmpty	False	statProfileNames	Null	filterList	No
opType	LOCK	fieldspec	epcBank	lockOperation	LOCK

CCSpec3					
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
Reader List	1 Reader	includeSpecInReports	False	startTriggerList	Null
duration	0	repeatPeriod	0	stopTriggerList	Null
noNewTagsInterval	0	tagsProcessedCount	1	afterError	False
reportIfEmpty	False	statProfileNames	Null	filterList	No
opType	LOCK	fieldspec	epcBank	lockOperation	UNLOCK

CCSpec4					
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
Reader List	1 Reader	includeSpecInReports	False	startTriggerList	Null
duration	0	repeatPeriod	0	stopTriggerList	Null
noNewTagsInterval	0	tagsProcessedCount	1	afterError	False
reportIfEmpty	False	statProfileNames	Null	filterList	No
opType	LOCK	fieldspec	epc	lockOperation	LOCK

- A Gen2 tag is placed in the reader field

Step	Step description	Expected results
1	A subscriber invokes the subscribe method to subscribe to the CCSpec1 providing its HTTP Notification URI	A user is subscribed to the CCSpec1 in the subscribe invocation and the associated command cycle is activated. Step 2 provides verification.
2	Invoke the getSubscribers method to verify that the subscribe method succeeded.	getSubscriber returns the list of notification URIs. The notification URI from step 1 should be in the list. Verify that the correct CCReports are being received at the notification URI per the boundary condition specified in the CCSpec1.
3	The subscriber un-suscribes CCSpec1 by invoking the ndefined t method using its HTTP Notification URI as a parameter.	The user is successfully unsubscribed.
4	The subscriber invokes the subscribe method to subscribe to CCSpec2 providing HTTP Notification URI.	The user is subscribed to the CCSpec2 in the subscribe invocation. Step 5 provides verification.
5	Invoke the getSubscriber method to verify that the subscribe method succeeded.	getSubscriber returns the list of notification URIs. The notification URI from step 4 should be in the list. Verify that the correct CCReports are being received at the notification URI per the boundary condition specified in the CCSpec2.
6	The subscriber un-subscribes CCSpec2 by invoking the unsubscribe method using its HTTP Notification URI as a parameter.	The user is successfully unsubscribed.
7	The subscriber invokes the subscribe method to subscribe to the CCSpec1 providing its HTTP Notification URI.	The user is subscribed to the CCSpec1 in the subscribe invocation. Step 8 provides verification.
8	Invoke the getSubscriber method to verify that the subscribe method succeeded.	getSubscriber returns the list of notification URIs. The notification URI from step 7 should be in the list. Verify that the correct CCReports are being received at the notification URI per the boundary condition specified in the CCSpec1. Verify that statusCode in tagReports is PERMISSION_ERROR.
9	The subscriber un-subscribes CCSpec1 by invoking the unsubscribe method using its HTTP Notification URI as a parameter.	The user is successfully unsubscribed.
10	The subscriber invokes the subscribe method to subscribe to the CCSpec3 providing its HTTP Notification URI	The user is subscribed to the CCSpec3 in the subscribe invocation. Step 11 provides verification.
11	Invoke the getSubscriber method to verify that the subscribe method succeeded.	getSubscriber returns the list of notification URIs. The notification URI from step 10 should be in the list. Verify that the correct CCReports are being received at the notification URI per the boundary condition specified in the CCSpec3.

12	The subscriber un-subscribes CCSpec3 by invoking the unsubscribe method using its HTTP Notification URI as a parameter.	The user is successfully unsubscribed.
13	The subscriber invokes the subscribe method to subscribe to the CCSpec1 providing its HTTP Notification URI.	The user is subscribed to the CCSpec1 in the subscribe invocation. Step 14 provides verification.
14	Invoke the getSubscriber method to verify that the subscribe method succeeded.	getSubscriber returns the list of notification URIs. The notification URI from step 13 should be in the list. Verify that the correct CCReports are being received at the notification URI per the boundary condition specified in the CCSpec1.
15	The subscriber un-subscribes CCSpec1 by invoking the unsubscribe method using its HTTP Notification URI as a parameter.	The user is successfully unsubscribed.
16	The subscriber invokes the subscribe method to subscribe to the CCSpec4 providing its HTTP Notification URI	The user is successfully unsubscribed.
17	Invoke the getSubscriber method to verify that the subscribe method succeeded.	getSubscriber returns the list of notification URIs. The notification URI from step 16 should be in the list. Verify that the correct CCReports are being received at the notification URI per the boundary condition specified in the CCSpec4. The "Operation not possible" error should be speicified in the report since the epc field cannot be lock.
18	The subscriber un-subscribes CCSpec4 by invoking the unsubscribe method using its HTTP Notification URI as a parameter.	The user is successfully unsubscribed.

581

582 15.6 TCR-W6 – Poll, Writing API

583

Poll, Writing API
TPId: TCR-W6
Requirement Purpose: This Test Case confirms that the invocation of poll method can provide a valid CCSpecName to an ALE implementation and return a CCReports consistent with the parameters set in the CCSpec and within the boundary conditions. **Requirements Tested:** GM1, GM13, GM14, GM15, GM16, GM18, WM1, WM2, WM6, WM9, WM11, WM14, WM16, WM17, WM19, WM21, WM38, WM44, WM45, WM46, WM47, WM48, WM49, WM50, WM51, WM52

Pre-test conditions:

- Two Valid CCSpecs have been defined: A and B
- Spec A: A valid CCSpec has been defined with a repeatPeriod of M seconds, a duration of N seconds where M > N
- Spec B: A valid CCSpec has been defined with a repeatPeriod of M seconds, a duration of N seconds where M < N as shown below:

CCSpec					
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
Reader List	1 Reader	includeSpecInReports	False	startTriggerList	Null
duration	N Sec	repeatPeriod	M sec	stopTriggerList	Null
noNewTagsInterval	0	tagsProcessedCount	0	afterError	False
reportIfEmpty	False	statProfileNames	Null	filterList	No
opType	READ	fieldspec	epc	dataspec	Null

Step	Step description	Expected results
1	Place a tag set in the reader field	
2	Invoke the poll using the name of spec A (M > N)	After N seconds, a CCReports that conforms to the ALE XSD should be returned listing all the tags in the reader field.
3	Remove all the tags from the reader field.	
4	Invoke the poll using the name of spec A (M > N)	After N seconds, a CCReports that conforms to the ALE XSD and that contains zero CCReport instances should be returned even though reportIfEmpty is false.
5	Place a tag set in the reader field	
6	Invoke the poll using the name of spec B (M < N)	After N seconds, a CCReports that conforms to the ALE XSD should be returned listing all the tags in the reader field.
7	Remove all the tags from the reader field.	
8	Invoke the poll using the name of spec B (M < N)	After N seconds, a CCReports that conforms to the ALE XSD and that contains zero CCReport instances should be returned even though reportIfEmpty is false.

584

585 15.7 TCR-W7 – Poll, Writing API

586

Poll, Writing API

TPId: TCR-W7

Requirement Purpose: This Test Case confirms that the invocation of two poll methods that use the same CCSpec but specify different parameters, the ALE implementation SHALL satisfy the second poll by a initiating a new command cycle rather than sharing the results of the first, as though the second poll were of a different CCSpec..
Requirements Tested: GM1, GM19, GM20, WM1, WM2, WM6, WM9, WM11, WM14, WM16, WM17, WM19, WM21, WM38, WM44, WM45, WM46, WM47, WM48, WM49, WM50, WM51, WM52

Pre-test conditions:

- A Valid CCSpecs has been defined

CCSpec					
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
Reader List	1 Reader	includeSpecInReports	False	startTriggerList	Null
duration	N Sec	repeatPeriod	N sec	stopTriggerList	Null
noNewTagsInterval	0	tagsProcessedCount	0	afterError	False
reportIfEmpty	False	statProfileNames	Null	filterList	No
opType	READ	fieldspec	epc	dataspec	Null

Step	Step description	Expected results
1	Place a tag set in the reader field	
2	Invoke a poll method setting the *params* parameter to a valid value.	After N seconds, a CCReports that conforms to the ALE XSD and that contains a CCReport instance should be returned.
3	Wait N/4 seconds	
4	Invoke the second poll method setting the *params* parameter to a valid value that is different from the *params* value from the first poll.	N seconds after the second poll is issued, a second CCReports that conforms to the ALE XSD and that contains a CCReport instance should be returned.
5	Invoke a third poll method without params.	N seconds after the third poll is issued, a CCReports that conforms to the ALE XSD and that contains a CCReport instance should be returned..
6	Wait N/4 seconds	
7	Invoke a fourth poll method without params	The fourth poll should use the same command cycle as the third poll so it should return at the same time as the third poll. A CCReports that conforms to the ALE XSD and that contains a CCReport instance should be returned..

587

588 15.8 TCR-W8 – Immediate, Writing API

589

Immediate, Writing API					
TPId: TCR-W8					
Requirement Purpose: This Test Case confirms that the invocation of the immediate method can provide a valid CCSpec to an ALE implementation and return CCReports consistent with the parameters set in the CCSpec and within the boundary conditions. The test also verifies the inclusion of a CCSpec in CCReports. **Requirements Tested:** GM1, GM21, GM22, GM23, GM24, GM25, WM1, WM2, WM6, WM7, WM9, WM11, WM14, WM16, WM17, WM19, WM21, WM38, WM44, WM45, WM46, WM47, WM48, WM49, WM50, WM51, WM52					
Pre-test conditions:					
CCSpec					
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
Reader List	1 Reader	includeSpecInReports	True	startTriggerList	Null
duration	N Sec	repeatPeriod	M sec	stopTriggerList	Null
noNewTagsInterval	0	tagsProcessedCount	0	afterError	False
reportIfEmpty	False	statProfileNames	Null	filterList	No
opType	READ	fieldspec	epc	dataspec	Null

Step	Step description	Expected results
1	Place a tag set in the reader field. Keeps the tags in the reader field for a time greater than the N seconds once the immediate method is issued.	
2	Invoke the immediate method using a valid CCSpec as specified in the pre-test conditions.	Ensure no CCReports is returned prior to N seconds. Ensure a CCReports that conforms to the ALE XSD is returned immediately after N seconds has past. Confirm the CCReports contains tag EPCs in Tag format and the CCSpec appears in the CCReports.

590

591 15.9 TCR-W9 – Using startTriggerList and stopTriggerList, 592 Writing API

593

Using startTriggerList and stopTriggerList, Writing API
TPId: TCR-W9
Requirement Purpose: This Test Case confirms the following features of the CCSpec: startTriggerList and stopTriggerList. **Requirements Tested:** GM1, WM1, WM6, WM9, WM11, WM14, WM16, WM17, WM19, WM21, WM38, WM44, WM45, WM46, WM47, WM48, WM49, WM50, WM51, WM52

Pre-test conditions:

- There are no users subscribed to the CCSpec.
- A valid CCSpec has been defined as shown

CCSpec					
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
Reader List	1 Reader	includeSpecInReports	False	startTriggerList	Yes
duration	N Sec	repeatPeriod	M sec	stopTriggerList	Yes
noNewTagsInterval	0	tagsProcessedCount	0	afterError	False
reportIfEmpty	False	statProfileNames	Null	filterList	No
opType	READ	fieldspec	epc	dataspec	Null

- StartTriggerList contains startTriggers URI1 and URI2. StopTriggerList contains stopTriggers URI3 and URI4.

Step	Step description	Expected results
1	Invoke the poll method to activate the CCSpec.	None
2	Move a set of tags into the reader field and trip the start trigger URI1.	The Command Cycle should begin
3	Trip the stop trigger URI3 after a sufficient time has passed for all the tags to have been written and reported to the ALE implementation.	A CCReports that conforms to the ALE XSD should be retuned by the poll command issued in step 1. The CCReports should contain a CCReport that contains the tag EPCs of those tags place in the reader field. The CCReports should also contain startTrigger URI1 as initiationCondition and stopTrigger URI3 as terminationCondition.
4	Invoke the poll method to activate the CCSpec.	None
5	Move a set of tags into the reader field and trip the start trigger URI2.	The Command Cycle should begin
6	Trip the stop trigger URI4 after a sufficient time has passed for all the tags to have been written and reported to the ALE implementation.	A CCReports that conforms to the ALE XSD should be retuned by the poll command issued in step 1. The CCReports should contain a CCReport that contains the tag EPCs of those tags place in the reader field. The CCReports should also contain startTrigger URI2 as initiationCondition and stopTrigger URI4 as terminationCondition.

594
595

15.10 TCR-W10 – Subscribe and Unsubscribe for KILL operation, Wrting API

Subscribe and Unsubscribe for KILL operation, Wrting API
TPId: TCR-W10
Requirement Purpose: This Test Case confirms that clients can subscribe and unsubscribe to CCSpecs that have been correctly defined and the notification URIs used conform to the ALE1.1 standard. Functionality of the field tagsProcessedCount is also verified. **Requirements Tested:** GM1, WM1, WM6, WM9, WM11, WM14, WM15, WM17, WM19, WM21, WM38, WM39, WM42, WM44, WM45, WM46, WM47, WM48, WM49, WM50, WM51, WM52, WM75

Pre-test conditions:

- Two valid CCSpecs have been defined as follows:

CCSpec1					
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
Reader List	1 Reader	includeSpecInReports	False	startTriggerList	Null
duration	0	repeatPeriod	0	stopTriggerList	Null
noNewTagsInterval	0	tagsProcessedCount	1	afterError	False
reportIfEmpty	False	statProfileNames	Null	filterList	No
opType	READ	fieldspec	epc	dataSpec	NULL

CCSpec2					
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
Reader List	1 Reader	includeSpecInReports	False	startTriggerList	Null
duration	0	repeatPeriod	0	stopTriggerList	Null
noNewTagsInterval	0	tagsProcessedCount	1	afterError	False
reportIfEmpty	False	statProfileNames	Null	filterList	No
opType	KILL	fieldspec	NULL	dataSpec	Kill Password

- A Gen2 tag with given Kill Password is placed in the reader field

Step	Step description	Expected results
1	A subscriber invokes the subscribe method to subscribe to the CCSpec1 providing its HTTP Notification URI	A user is subscribed to the CCSpec1 in the subscribe invocation and the associated command cycle is activated. Step 2 provides verification.
2	Invoke the getSubscribers method to verify that the subscribe method succeeded.	getSubscriber returns the list of notification URIs. The notification URI from step 1 should be in the list. Verify that the correct CCReports are being received at the notification URI per the boundary condition specified in the CCSpec1.

3	The subscriber un-suscribes CCSpec1 by invoking the "undefined" method using its HTTP Notification URI as a parameter.	The user is successfully unsubscribed.
4	The subscriber invokes the subscribe method to subscribe to CCSpec2 providing HTTP Notification URI.	The user is subscribed to the CCSpec2 in the subscribe invocation. Step 5 provides verification.
5	Invoke the getSubscriber method to verify that the subscribe method succeeded.	getSubscriber returns the list of notification URIs. The notification URI from step 4 should be in the list. Verify that the correct CCReports are being received at the notification URI per the boundary condition specified in the CCSpec2.
6	The subscriber un-subscribes CCSpec2 by invoking the unsubscribe method using its HTTP Notification URI as a parameter.	The user is successfully unsubscribed.

599
600

601 15.11 TCR-W11 – Using EPCCache, Writing API

602

Using EPCCache, Writing API

TPId: TCR-W11

Requirement Purpose: This Test Case confirms that clients can subscribe and unsubscribe to CCSpecs that have been correctly defined and the notification URIs used conform to the ALE1.1 standard. Functionality of the field tagsProcessedCount is also verified.
Requirements Tested: GM1, WM1, WM6, WM9, WM11, WM14, WM17, WM19, WM21, WM38, WM39, WM44, WM45, WM46, WM47, WM48, WM49, WM50, WM51, WM52, WM53, WM54, WM55, WM56, WM59, WM60

Pre-test conditions:

- A valid CCSpec specified as follows should be used for step 10a.:

CCSpec					
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
Reader List	1 Reader	includeSpecInReports	False	startTriggerList	Null
duration	0	repeatPeriod	0	stopTriggerList	Null
noNewTagsInterval	0	tagsProcessedCount	1	afterError	False
reportIfEmpty	False	statProfileNames	Null	filterList	No
opType	WRITE	fieldspec	epc	opDataSpecType	CACHE
data	*name of the EPCCache defined in step 1*				

- A Gen2 tagset is placed in the reader field

Step	Step description	Expected results
1	Invoke a defineEPCCache method with a null EPCCacheSpec and an EPCPatternList (containing *patterns1* list) as replenishment.	The EPCCache is defined.
2	Invoke a getEPCCacheNames method.	Verify that getEPCCacheNames returns the name of the EPCCache defined in step 1.
3	Invoke a getEPCCache method with the name of the EPCCache defined in step 1.	Verify that getEPCCache returns the EPCCacheSpec used in step 1.
4	Invoke a getEPCCacheContents method with the name of the EPCCache defined in step 1.	Verify that EPCPatternList containing *patterns1* list is returned.
5	Invoke a replenishEPCCache method with the name of the EPCCache defined in step 1 and replenishment (EPCPatternList containing *patterns2* list).	The EPCCache is replenished.
6	Invoke a getEPCCacheContents method with the name of the EPCCache defined in step 1.	Verify that EPCPatternList containing *patterns1* and *patterns2* lists is returned.
7	Invoke a depleteEPCCache method to remove from the contents of the EPCCache defined in step 1.	The EPCCache is depleted.
8	Invoke a replenishEPCCache method with the name of the EPCCache defined in step 1 and replenishment (EPCPatternList containing *patterns1* list).	The EPCCache is replenished with *patterns1*
9	Invoke a getEPCCacheContents method with the name of the EPCCache defined in step 1.	Verify that EPCPatternList containing *patterns1* list is returned.
10a.	Define the above mentioned CCSpec.	The CCSpec is defined.
10b.	A subscriber invokes the subscribe method to subscribe to the CCSpec providing its HTTP Notification URI	A user is subscribed to the CCSpec in the subscribe invocation and the associated command cycle is activated.
11	Wait for CCReports	Verify that the correct CCReports are being received at the notification URI per the boundary condition specified in the CCSpec.
12a.	Undefine the above mentioned CCSpec.	The CCSpec is undefined.
12b.	Invoke an undefineEPCCache method to ndefined the EPCCache defined in step 1.	The EPCCache is undefined.
13	Invoke a getEPCCacheNames method.	An empty list of names is returned.

603

604 15.12 TCR-W12 – Using Association Table, Writing API

605

Using Association Table, Writing API

TPId: TCR-W12

Requirement Purpose: This Test Case confirms that clients can subscribe and unsubscribe to CCSpecs that have been correctly defined and the notification URIs used conform to the ALE1.1 standard.
Requirements Tested: GM1, WM1, WM6, WM9, WM11, WM14, WM17, WM19, WM21, WM38, WM39, WM44, WM45, WM46, WM47, WM48, WM49, WM50, WM51, WM52, WM61, WM62, WM64, WM66, WM67

Pre-test conditions:

- A valid CCSpec specified as follows should be used for step 1b:

AssoTableSpec					
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
datatype	epc	Format	epc-hex	Entries	Valid AssoTableEntry

CCSpec					
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
Reader List	1 Reader	includeSpecInReports	False	startTriggerList	Null
duration	0	repeatPeriod	0	stopTriggerList	Null
noNewTagsInterval	0	tagsProcessedCount	1	afterError	False
reportIfEmpty	False	statProfileNames	Null	filterList	No
opType	WRITE	fieldspec	epc	opDataSpecType	ASSOCIATION
data	*name of the Association Table defined in step 1*				

- A Gen2 tagset is placed in the reader field

Step	Step description	Expected results
1a.	Invoke a defineAssocTable method with AssoTableSpec.	The association table is defined.
1b.	Define the above mentioned CCSpec.	The CCSpec is defined.
2	Invoke getAssocTableNames method.	The name of the association table defined in step 1 is returned.
3	Invoke getAssocTable method using the name of the association table defined in step 1.	The AssoTableSpec is returned.
4	A subscriber invokes the subscribe method to subscribe to the CCSpec providing its HTTP Notification URI	A user is subscribed to the CCSpec in the subscribe invocation and the associated command cycle is activated.
5		Verify that the correct CCReports are being received at the notification URI per the boundary condition specified in the CCSpec.

6a.	Undefine the CCSpec defined in Step 1b.	The CCSpec is undefined.
6b.	Invoke an undefineAssocTable method to undefined the association table defined in step 1.	The association table is undefined.
7	Invoke a getAssocTableNames method.	An empty list of names is returned.

606 ## 15.13 TCR-W13 – Using RNG, Writing API

607

Using RNG, Writing API

TPId: TCR-W13

Requirement Purpose: This Test Case confirms that clients can subscribe and unsubscribe to CCSpecs that have been correctly defined and the notification URIs used conform to the ALE1.1 standard.
Requirements Tested: GM1, WM1, WM6, WM9, WM11, WM14, WM17, WM19, WM21, WM38, WM39, WM44, WM45, WM46, WM47, WM48, WM49, WM50, WM51, WM52, WM53, WM68, WM69, WM71, WM72, WM73, WM74

Pre-test conditions:

- A valid CCSpec specified as follows should be used for step 1b:

CCSpec					
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
Reader List	1 Reader	includeSpecInReports	False	startTriggerList	Null
duration	0	repeatPeriod	0	stopTriggerList	Null
noNewTagsInterval	0	tagsProcessedCount	1	afterError	False
reportIfEmpty	False	statProfileNames	Null	filterList	No
opType	WRITE	fieldspec	@1.96.32	opDataSpecType	RANDOM
data	*name of the RNG defined in step 1*				

- A Gen2 tagset is placed in the reader field

Step	Step description	Expected results
1a.	Invoke a defineRNG method with a RNGSpec.	The RNG is defined.
1b.	Define the above mentioned CCSpec.	The CCSpec is defined.
2	Invoke getRNGNames method.	The name of the RNG defined in step 1 is returned.
3	A subscriber invokes the subscribe method to subscribe to the CCSpec providing its HTTP Notification URI	A user is subscribed to the CCSpec in the subscribe invocation and the associated command cycle is activated.
4		Verify that the correct CCReports are being received at the notification URI per the boundary condition specified in the CCSpec.
5a.	Undefine the CCSpec defined in Step 1b.	The CCSpec is undefined.

5b.	Invoke an undefineRNG method to ndefined the RNG defined in step 1.	The RNG is undefined.

608

609 15.14TCR-W14 – Memory Banks, Writing API

610

Memory Banks, Writing API
TPId: TCR-W14
Requirement Purpose: This Test Case demonstrates demonstrates writing to all Gen memory banks. Both fix and the variable fields wo;; be tested. Writing features by writing data to a tag's user memory bank. **Requirements Tested:** GM1, GM26, GM27, GM29, GM33, GM35, GM36, GM37, GM38, GM39, GM40, GM42, GM43. GM44, GM46, GM47, GM48, GM50, GM50, GM51, GM52, GM54, GM55, GM56, GM57, GM59, GM60, GM61, GM63, GM65, GM66, GM67, GM68, GM69, GM71, GM72, GM73, GM74, GM75, GM77, GM78, GM79, GM80, GM81, GM82, GM84, GM85, GM86, GM87, GM88, GM89, GM90, GM91, GM93, GM94, GM95, GM96, GM97, GM98, GM99, GM100, GM101, GM102, GM103, GM104, GM105, GM106, GM107, GM108, GM109, GM110, GM111, GM112, WM1, WM6, WM9, WM11, WM12, WM14, WM17, WM19, WM20, WM21, WM23, WM24, WM25, WM25, WM27, WM28, WM30, WM31, WM32, WM34, WM35, WM36, WM38, WM39, WM44, WM45, WM46, WM47, WM48, WM49, WM50, WM51, WM52, WM58

Pre-test conditions:

- A tag with user memory
- CCSpec and corresponding CCOpSpec lists for the test:

Note: Steps 14 through 19 are optional is the implementation does not support variable fields.

CCSpec					
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
Reader List	1 Reader	includeSpecInReports	False	startTriggerList	Null
duration	N Sec	repeatPeriod	M sec	stopTriggerList	Null
noNewTagsInterval	0	tagsProcessedCount	0	afterError	False
reportIfEmpty	True	statProfileNames	Null	filterList	No

CmdSpec → CCOpSpec List A		
OpType	**FieldSpec**	**DataSpec**
WRITE	@0.32	LITERAL; valid hexadecimal value
WRITE	@0.32.32	LITERAL; valid hexadecimal value
WRITE	@1.96.32	LITERAL; valid EPC
WRITE	@2.32	LITERAL; valid hexadecimal value
WRITE	@3.32	LITERAL; valid hexadecimal value

CmdSpec → CCOpSpec List B		
OpType	**FieldSpec**	**DataSpec**
READ	@0.32	
READ	@0.32.32	
READ	@1.96.32	
READ	@2.32	
READ	@3.32	

CmdSpec → CCOpSpec List C		
OpType	**FieldSpec**	**DataSpec**
WRITE	epcBank	LITERAL; valid hexadecimal bits value
WRITE	tidBank	LITERAL; valid hexadecimal bits value
WRITE	userBank	LITERAL; valid hexadecimal bits value
WRITE	killPwd	LITERAL; valid hexadecimal value
Write	accessPwd	LITERAL; valid hexadecimal value

CmdSpec → CCOpSpec List D		
OpType	**FieldSpec**	**DataSpec**
READ	epcBank	
READ	tidBank	
READ	userBank	
READ	killPwd	
READ	accessPwd	

CmdSpec → CCOpSpec List E		
OpType	**FieldSpec**	**DataSpec**
WRITE	afi	LITERAL; valid hexadecimal value

CmdSpec → CCOpSpec List F		
OpType	**FieldSpec**	**DataSpec**
READ	afi	

CmdSpec → CCOpSpec List G		
OpType	**FieldSpec**	**DataSpec**
WRITE	nsi	LITERAL; valid hexadecimal value

CmdSpec → CCOpSpec List H		
OpType	**FieldSpec**	**DataSpec**
READ	nsi	

CCOpSpec List I		
OpType	**FieldSpec**	**DataSpec**
INITIALIZE	userBank	urn:epcglobal:ale:init:iso15962:x0C
ADD	@3.urn:oid:1.0.15961.12.4	CID
ADD	@3.urn:oid.1.0.15961.12.5	CIDDFWLAX
ADD	@3.urn:oid:1.0.15962.12.6	AA353208AUGDFW
ADD	@3.urn:oid:1.0.15962.12.7	DOE,JOHN

CCOpSpec List J		
OpType	**FieldSpec**	**DataSpec**
READ	@3.urn:oid:1.0.15961.12.4	
READ	@3.urn:oid.1.0.15961.12.5	
READ	@3.urn:oid:1.0.15962.12.6	
READ	@3.urn:oid:1.0.15962.12.7	

CCOpSpec List K		
OpType	**FieldSpec**	**DataSpec**
CHECK	userBank	urn:epcglobal:ale:check:iso15962
WRITE	@3.urn:oid:1.0.15962.12.7	DEER,JOHN

CCOpSpec List L		
OpType	**FieldSpec**	**DataSpec**
WRITE	@3.urn:oid:1.0.15962.12.7	DEER,JOHN

CCOpSpec List M		
OpType	**FieldSpec**	**DataSpec**
DELETE	@3.urn:oid:1.0.15962.12.7	

CCOpSpec List N		
OpType	**FieldSpec**	**DataSpec**
READ	@3.urn:oid:1.0.15962.12.7	

CmdSpec → CCOpSpec List O		
OpType	**FieldSpec**	**DataSpec**
WRITE	afi	LITERAL; valid hexadecimal value greater than 8 bits in length (> 256)

Step	Step description	Expected results
1	Place a single Gen 2 tag with epc, user, TID and reserve memory in the reader's field. The fields should be cleared.	
2	Invoke the immediate method using the CCSpec with CCOpSpec List A as specified in the pre-test conditions	After N seconds, a CCReports that conforms to the ALE XSD should be returned indicating that the CCOpSpecs have been executed successfully.
3	Invoke the immediate method using the CCSpec with CCOpSpec List B as specified in the pre-test conditions	After N seconds, a CCReports that conforms to the ALE XSD should be returned indicating that the CCOpSpecs have been executed successfully and contain the content of the reserve, epc, TID and user memory banks as written in step 2
4	Place a single Gen 2 tag with epc, user, TID and reserve memory in the reader's field. The fields should be cleared.	
5	Invoke the immediate method using the CCSpec with CCOpSpec List C as specified in the pre-test conditions	After N seconds, a CCReports that conforms to the ALE XSD should be returned indicating that the CCOpSpecs have been executed successfully.

6	Invoke the immediate method using the CCSpec with CCOpSpec List D as specified in the pre-test conditions	After N seconds, a CCReports that conforms to the ALE XSD should be returned indicating that the CCOpSpecs have been executed successfully and contain the content of the reserve, epc, TID and user memory banks as written in step 5. Note: The "operation not possible" condition could be returned for epcBank, tidBank and userBank if the ALE implementation does not support reading to the end of the memory bank. In this case no results for these fields will be returned.
7	Place a single Gen 2 tag with epc, user, TID and reserve memory in the reader's field.	
8	Invoke the immediate method using the CCSpec with CCOpSpec List E as specified in the pre-test conditions	After N seconds, a CCReports that conforms to the ALE XSD should be returned indicating that the CCOpSpecs have been executed successfully.
9	Invoke the immediate method using the CCSpec with CCOpSpec List F as specified in the pre-test conditions	After N seconds, a CCReports that conforms to the ALE XSD should be returned indicating that the CCOpSpecs have been executed successfully and contains the afi field written in step 8.
10	Place a single Gen 2 tag with epc, user, TID and reserve memory in the reader's field.	
11	Invoke the immediate method using the CCSpec with CCOpSpec List G as specified in the pre-test conditions	After N seconds, a CCReports that conforms to the ALE XSD should be returned indicating that the CCOpSpecs have been executed successfully.
12	Invoke the immediate method using the CCSpec with CCOpSpec List H as specified in the pre-test conditions	After N seconds, a CCReports that conforms to the ALE XSD should be returned indicating that the CCOpSpecs have been executed successfully and contains the nsi field written in step 1`.
13	(optional) Place a single Gen 2 tag with epc, user, TID and reserve memory in the reader's field. The fields should be cleared	
14	(optional) Invoke the immediate method using the CCSpec with CCOpSpec List I as specified in the pre-test conditions.	After N seconds, a CCReports that conforms to the ALE XSD should be returned indicating that the CCOpSpecs have been executed successfully. Alternatively, if an implementation does not support variable fields, an "operation not possible" condition should be raised.

15	(optional) Invoke the immediate method using the CCSpec with CCOpSpec List J as specified in the pre-test conditions.	After N seconds, a CCReports that conforms to the ALE XSD should be returned indicating that the CCOpSpecs have been executed successfully.. The report should contain the values written in step 14. Alternatively, if an implementation does not support variable fields, an "operation not possible" condition should be raised.
16	(optional) Invoke the immediate method using the CCSpec with CCOpSpec List K as specified in the pre-test conditions.	After N seconds, a CCReports that conforms to the ALE XSD should be returned indicating that the CCOpSpecs have been executed successfully. Alternatively, if an implementation does not support variable fields, an "operation not possible" condition should be raised.
17	(optional) Invoke the immediate method using the CCSpec with CCOpSpec List L as specified in the pre-test conditions	Use the report resulting from Step-17 to verify that the data has been correctly re-written to the tag's user memory. Alternatively, if an implementation does not support variable fields, an "operation not possible" condition should be raised.
18	(optional) Invoke the immediate method using the CCSpec with CCOpSpec List M as specified in the pre-test conditions.	After N seconds, a CCReports that conforms to the ALE XSD should be returned indicating that the CCOpSpecs have been executed successfully. Alternatively, if an implementation does not support variable fields, an "operation not possible" condition should be raised.
19	(optional) Invoke the immediate method using the CCSpec with CCOpSpec List N as specified in the pre-test conditions	Use the report resulting from Step-19 to verify that the data has been correctly deleted from the tag's user memory. Alternatively, if an implementation does not support variable fields, an "operation not possible" condition should be raised.
20	Invoke the immediate method using the CCSpec with CCOpSpec List O as specified in the pre-test conditions	An CCReport should be returned that contains an "out of range" condition. The CCReport should have a CCStatus of "OUT_OF_RANGE_ERROR"

15.15 TCR-W15 – Initiation and Termination Conditions for Undefining a CCSpec during Active Poll, Writing API

Initiation and Termination Conditions for Undefining an Active Poll, Reading API
TPId: TCR-W15

Requirement Purpose: This Test Case Requirement verifies that the initiation and termination conditions are properly filled when an active CCSpec, as a result of a Poll call, is undefined. **Requirements Tested:** GM1, GM17, WM1, WM14, WM17, WM19, WM21		
Pre-test conditions: • A valid CCSpec has been defined with repeatPeriod = 0 sec, duration = N sec.		
Step	**Step description**	**Expected results**
1	Invoke the poll method using the CCSpec defined in pre-test condition.	Poll call is outstanding.
2	Wait for N/2 seconds	-
3	Invoke ndefined method to ndefined the CCSpec defined in step 1.	Verify that an CCReports is received right after the ndefined call with InitiationCondition = REQUESTED and TerminationCondition = UNDEFINE.

614

615 ## 15.16– XML Vendor Extension Validaion

XML Vendor Extension Validation		
TPId: TCR-W16		
Requirement Purpose: This Test Case confirms that vendor extensions to the CCSpec, CCReports, EPCCacheSpec, AssociationTableSpec and RNGSpec have been added in accordance with the rules set forth in the ALE 1.1 specification. This TCR is opational. This TCR only needs to be executed for implementation that have vendor extension. **Requirements Tested:** XM1, XM2, XM3, XM4, XM5, XM6, XM8, XM9, XM12		
Pre-test conditions: • The vendor has submitted XML files containing instances of CCSpecs, CCReports, EPCCacheSpec, AssociationTableSpec and RNGSpec that contain the the vendor extensions or the vendor's XSD for the Writing API or appropriate documentation confirming the vendor is the owner of the namespace used for the vendor extensions. • The vendor has provided its XSD files so they can be inspected to ensure that elements, attributes and extensions have not be added in places not allowed by the specification.		
Step	**Step description**	**Expected results**
1	Examine the XML documents, XSD documents or other documentation submitted by the vendor to verify the vendor is the owning authority for the name space used for all vendor attribute and element extensions.	Confirm (by design)
2	Validate the XML CCSpec, CCReports, EPCCacheSpec, AssociationTableSpec and RNGSpec instance documents received in TCR-W1 through W15 against the ALE 1.1 Writing API XSD. (See section 13.4)	The XML documents should validate successfully.

3	Inspect the vendor XSDs to ensure that elements, attributes and extensions have not be added in places not allowed by the ALE 1.1 specification.	There are no elements, attributes and extensions added in the vendor's XSDs in places not allowed by the ALE 1.1 specificatoin.

616

617

618

16 Tag Memory Specification API

16.1 TCR-T1 – Get Version, Tag Memory API

Get Version, Tag Memory API		
TPId: TCR-T1		
Test Purpose: This Test Case confirms the proper functions of the methods of the Tag Memory API that return the ALE standard version and the vendor version for the implementation under test. The return of correct version numbers also confirms the correct implementation is being tested. **Requirements Tested :** GM1, GM2, GM3, GM4, GM5, TM2		
Pre-test conditions: • None		
Step	**Step description**	**Expected results**
1	Invoke the getStandardVersion method of Tag Memory API	• Confirm the string "1.1" is returned. • Confirm the result returned by this method only pertain to the API to the Tag Memory API
2	Invoke the getVendorVersion method of the Tag Memory API	• Confirm that either an empty string or a string conforming to a proper URI is returned. • Confirm the vendor is the owning authority of the URI if the returned string is not empty (by Design) • Confirm the result returned by this method only pertain to the API to the Tag Memory API.

16.2 TCR-T2 – Defining, Un-defining, Retrieving TMSpecs, Tag Memory API

Defining, Un-defining, Retrieving TMSpecs, Tag Memory API		
TPId: TCR-T2		
Test Purpose: This Test Case confirms that a valid Tag Memory Spec can be defined and undefined. Further the defining and un-defining of the Tag Memory Spec can be verified with "ALETM" API methods getTMSpec and getTMSpecNames. **Requirements Tested** : GM1, TM1, TM2, TM5		
Pre-test conditions: • No TMSpec is defined		
Step	**Step description**	**Expected results**
1	Invoke the defineTMSpec method with a valid TMFixedFieldListSpec TMSpec	The ALETM implementation contains the TMSpec definition supplied in the define method. Steps 2 and 3 confirm the defining of the TMSpec.
2	Verify the TMSpec was defined by invoking the getTMSpecNames method	Verify that the name returned in the list is that of the TMSpec just defined

3	Invoke getTMSpec using the name of the defined TMSpec.	Verify that the TMSpec returned is the same as the one defined
4	Invoke the defineTMSpec method with a valid TMSpec	The ALETM implementation contains the TMSpec definition supplied in the define method.
5	Verify the TMSpecs were defined by invoking the getTMSpecNames method	Verify that the names returned in the list is that of the TMSpecs just defined
6	Repeat steps 1 to 5 with a valid TMVariableFieldListSpec TMSpec before proceeding to step 7	The ALETM implementation contains the TMSpec definition supplied in the define method. Steps 2 and 3 confirm the defining of the TMSpec.
7	Invoke undefineTMSpec to remove the TMSpec that was defined in step 1.	The ALETM implementation should no longer have the TMSpecs defined.
8	Verify that the TMSpecs in step 1 is undefined by invoking the getTMSpecNames method.	Verify that the list returned only contain the spec name defined in step 4
9	Invoke undefineTMSpec to remove the TMSpecs that was defined in step 4.	The ALETM implementation should no longer have the TMSpecs defined.
10	Verify that the TMSpec is undefined by invoking the getTMSpecNames method.	Verify that only the TMVariableFieldListSpec name is returned.
11	Repeat steps 7 to 10 with for TMVariableFieldListSpec TMSpec	Verify that the list returned is empty.

626

627 ## 16.3 TCR-T3 – Exceptions, Tag Memory API

Exceptions, Tag Memory API		
TPId: TCR-T3		
Test Purpose: This Test Case confirms that ALETM implementation will raise all exceptions as defined in the ALETM specification. **Requirements Tested :** GM1, TM2, TM4, TM7, TM9, TM12		
Pre-test conditions: • No TMSpec is defined. • Use TMFixedFieldListSpec TMSpecs for the first pass through steps 1 through 11.		
Step	**Step description**	**Expected results**
1	Invoke the defineTMSpec method with a valid TMSpec name = “TM1”.	The ALETM implementation contains the TMSpec definition supplied in the define method.
2	Invoke the defineTMSpec method with a valid TMSpec name = “TM1”.	Verify that the ALETM implementation raises a DuplicateNameException.
3	Invoke the defineTMSpec method with a valid TMSpec name = “TM2”.field parms are Bank = -1; length = -3; offset = -1. (This step is skipped for TMVariableFieldListSpec).	Verify that the ALETM implementation raises a TMSpecValidationException.
4	Invoke the undefineTMSpec method with name = “TM2”.	Verify that the ALETM implementation raises a NoSuchNameException

5	Invoke the getTMSpec method with name = “TM2”.	Verify that the ALETM implementation raises a NoSuchNameException
6	Invoke undefineTMSpec method with name = “TM1”	TMSpec will be successfully removed.
7	Invoke the defineTMSpec method with a valid TMSpec name = “TM3”. Field parms are Bank = 0; OID= ”urn:epcglobal:1.0.15961.12.11” (This step is skipped for TMFixedFieldListSpec).	Verify that the ALETM implementation raises a TMSpecValidationException.
8	Invoke the defineTMSpec method with a valid TMSpec of specName = “TM1” and fieldname = “symbol”.	The ALETM implementation contains the TMSpec definition supplied in the defineTMSpec method.
9	Invoke the defineTMSpec method with a valid TMSpec of specName = “TM2” and fieldname = “symbol”.	Verify that the ALETM implementation raises a TMSpecValidationException.
10	Invoke the defineTMSpec method with a valid TMSpec of specName = “TM2” and fieldname = any built-in fieldname in ALE1.1 specification section 6.1.	Verify that the ALETM implementation raises a TMSpecValidationException.
11	Invoke the defineTMSpec method with a valid TMSpec of specName = “TM2” and fieldname = “@symbol”.	Verify that the ALETM implementation raises a TMSpecValidationException.
12	Repeat steps 1 through 11 with a TMVariableFieldListSpec	

628

629 ## 16.4 TCR-T4 – Using Fixed Fieldnames defined with Tag Memory
630 API

Using Fieldnames defined with Tag Memory API
TPId: TCR-T4
Test Purpose: This Test Case confirms that a valid fixed fieldname defined with Tag Memory APIs can be usable in an ECSpec. **Requirements Tested :** GM1, TM1, TM2, TM5, TM8

Pre-test conditions:

- No TMSpec is defined.
- A valid ECSpec is defined as shown below:

TMFixedFieldSpec					
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
fieldname	PC	bank	1	length	16
offset	16	defaultDatatype	uint	defaultFormat	hex

ECSpec					
Parameter	Value	Parameter	Value	Parameter	Value
Reader List	1 Reader	repeatPeriod	M	startTrigger	omitted
duration	N	stableSetInterval	0	stopTrigger	omitted
Current	Yes	Additions	No	Deletions	No
includeCount	Yes	includeEPC	No	includeTag	Yes
reportIfEmpty	False	reportOnlyOnChange	False	includeSpecInReports	False
includeRawHex	No	includeRawDecimal	Yes	groupSpec	No
includePatterns	No	excludePatterns	Yes		
filterList	Yes	statProfileNames	No		

ECFilterListMember					
Parameter	Value	Parameter	Value	Parameter	Value
includeExclude	Include	fieldspec	ECFieldspec	patList	*valid pattern list*

ECFieldSpec					
Parameter	Value	Parameter	Value	Parameter	Value
Fieldname	PC	Datatype	omitted	format	omitted

- A Gen2 tagset is kept in the reader field

Step	Step description	Expected results
1	Invoke the defineTMSpec method with a valid TMSpec using TMFixedFieldSpec.	The ALETM implementation contains the TMSpec definition supplied in the define method. Steps 2 and 3 confirm the defining of the TMSpec.
2	Verify the TMSpec was defined by invoking the getTMSpecNames method.	Verify that the name returned in the list is that of the TMSpec just defined
3	Invoke the define method of the reading API to define the ECSpec	The ECSpec as specified in the pre-test conditions is defined.
4	Invoke the poll method of the reading API to activate the ECSpec and begin the event cycle.	The event cycle should have started.

5	Verify that after time N, when the duration expires, an ECReports is returned by the poll.	An ECReports that conforms to the ALE XSD should be returned by the poll in Step 4. It should include those tags from the tag set that matches the include filter.
6	Invoke ndefined of the reading API method to remove ECSpec.	The ALE implementation undefines the ECSPec.
7	Invoke undefineTMSpec to remove the TMSpec that was defined in step 1.	The ALETM implementation should no longer have the TMSpecs defined.
8	Invoke define method of the reading API to define the ECSpec at the ALE ndefined tion.	The ALE implementation raises ECSpecvalidationException.

631

632 16.5 TCR-T5 – Using Variable Fieldnames defined with Tag
633 Memory API

Using Fieldnames defined with Tag Memory API
TPId: TCR-T5
Test Purpose: This Test Case confirms that a valid variable fieldname defined with Tag Memory APIs can be usable in an ECSpec. **Requirements Tested :** GM1, TM2, TM6, T10, T11 **Note:** This test requirement is optional and should not be executed for implementations that only support fixed fileds. It should be executed for implementation that supports variable fields.

Pre-test conditions:

- No TMSpec is defined.
- A valid ECSpec is defined as shown below:

TMVariableFieldSpec					
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
fieldname	PC	bank	1	oid	urn:oid:1.0.15961.11.12

ECSpec					
Parameter	Value	Parameter	Value	Parameter	Value
Reader List	1 Reader	repeatPeriod	M	startTrigger	omitted
duration	N	stableSetInterval	0	stopTrigger	omitted
Current	Yes	Additions	No	Deletions	No
includeCount	Yes	includeEPC	No	includeTag	Yes
reportIfEmpty	False	reportOnlyOnChange	False	includeSpecInReports	False
includeRawHex	No	includeRawDecimal	Yes	groupSpec	No
includePatterns	No	excludePatterns	Yes		
filterList	Yes	statProfileNames	No		

ECFilterListMember					
Parameter	Value	Parameter	Value	Parameter	Value
includeExclude	Include	fieldspec	ECFieldspec	patList	*valid pattern list*

ECFieldSpec					
Parameter	Value	Parameter	Value	Parameter	Value
Fieldname	PC	Datatype	omitted	format	omitted

- A Gen2 tagset is kept in the reader field

Step	Step description	Expected results
1	Invoke the defineTMSpec method with a valid TMSpec using TMVariableFieldSpec.	The ALETM implementation contains the TMSpec definition supplied in the define method. Steps 2 and 3 confirm the defining of the TMSpec.
2	Verify the TMSpec was defined by invoking the getTMSpecNames method.	Verify that the name returned in the list is that of the TMSpec just defined
3	Invoke the define method of the reading API to define the ECSpec	The ECSpec as specified in the pre-test conditions is defined.
4	Invoke the poll method of the reading API to activate the ECSpec and begin the event cycle.	The event cycle should have started.

5	Verify that after time N, when the duration expires, an ECReports is returned by the poll.	An ECReports that conforms to the ALE XSD should be returned by the poll in Step 4. It should include those tags from the tag set that matches the include filter.
6	Invoke ndefined method of the reading API to remove ECSpec.	The ALE implementation undefines the ECSPec.
7	Invoke undefineTMSpec to remove the TMSpec that was defined in step 1.	The ALETM implementation should no longer have the TMSpecs defined.
8	Invoke define method of the reading API to define ECSpec at the ALE ndefined tion.	The ALE implementation raises ECSpecvalidationException.

634

635 ## 16.6 TCR-T6 – XML Vendor Extension Validaion

XML Vendor Extension Validation		
TPId: TCR-T6		
Requirement Purpose: This Test Case confirms that vendor extensions to the TMSpec have been added in accordance with the rules set forth in the ALE 1.1 specification. This TCR is opational. This TCR only needs to be executed for implementation that have vendor extension. **Requirements Tested:** XM1, XM2, XM3, XM4, XM5, XM6, XM8, XM9, XM12		
Pre-test conditions: • The vendor has submitted XML files containing instances of TCSpecs that contain the the vendor extensions or the vendor's XSD for the Tag Memory API or appropriate documentation confirming the vendor is the owner of the namespace used for the vendor extensions. • The vendor has provided its XSD files so they can be inspected to ensure that elements, attributes and extensions have not be added in places not allowed by the specification.		
Step	**Step description**	**Expected results**
1	Examine the XML documents, XSD documents or other documentation submitted by the vendor to verify the vendor is the owning authority for the name space used for all vendor attribute and element extensions.	Confirm (by design)
2	Validate the XML TMSpec instance documents received in TCR-T1 through T5 against the ALE 1.1 Tag Memory API XSD. (See section 13.4)	The XML documents should validate successfully.
3	Inspect the vendor XSDs to ensure that elements, attributes and extensions have not be added in places not allowed by the ALE 1.1 specification.	There are no elements, attributes and extensions added in the vendor's XSDs in places not allowed by the ALE 1.1 specificatoin.

636

637

17 Access Control API

17.1 TCR-A1 – Get Version, Access Control API

Get Version, Access Control API		
TPId: TCR-A1		
Requirement Purpose: This Test Case confirms the proper functions of the ALE methods of the Access Control API that return the ALE standard version and the vendor version for the ALE implementation under test. The return of correct version numbers also confirms the correct implementation is being tested. **Requirements Tested:** GM1, GM2, GM3, GM4, GM5, AM1, AM15		
Pre-test conditions: • None		
Step	**Step description**	**Expected results**
1	Invoke the getStandardVersion method of the Access Control API.	• Confirm the string "1.1" is returned. • Confirm the result returned by this method only pertain to the API to the Access Control API.
2	Invoke the getVendorVersion method of the Access Control API.	• Confirm that either an empty string or a string conforming to a proper URI is returned. • Confirm the vendor is the owning authority of the URI if the returned string is not empty (by Design) • Confirm the result returned by this method only pertain to the API to the Access Control API.

17.2 TCR-A2 – Supported Operations

Supported Operations		
TPId: TCR-A2		
Requirement Purpose: This Test Case confirms the proper function of the getSupportedOperations method and correct handling of unsupported operations. **Requirements Tested:** GM1, AM1, AM15, AM16, AM17, AM18, AM19, AM20, AM21		
Pre-test conditions: • None		
Step	**Step description**	**Expected results**
1	Invoke the getSupportedOperations method of the Access Control API.	• Confirm that at least the getStandardVersion, getVendorVersion, and getSupportedOperations are in the unordered list returned • Confirm the list of methods returns conform to the rules in Section 11.8

2	If an implementation does not support all the methods, test a subset of the methods not supported to ensure they raise an UnsupportedOperationExceptions	An unsupported exception should be raised
3	If the implementation supports an anonymous client identity, show documentation on how this is done.	Examine the provided document on anonymous client identity.
4	Show documentation on how at least one client establishes permission or out-of-band mechanism that is used to grant access to ordinary clients.	Examine the provided document to confirm the existence of the client or mechanism.

644

645 17.3 TCR-A3 – Using ClientIdentity, Roles and Permissions, 646 Access Control API

647

Using ClientIdentity, Roles and Permissions, Access Control API
TPId: TCR-A3
Requirement Purpose: This Test Case confirms valid AC Permissions used by other ALE implementations. **Requirements Tested:** GM1, AM1, AM2, AM4, AM6, AM10, AM11, AM12, AM13, AM14, AM15, RM3, WM4, TM3, LM8
NOTE: Certain combinations of permission- and role-related methods may raise UnsupportedOperationException instead of performing the expected function.

Pre-test conditions:

- Only the minimum number of ACPermission, ACRole and ACClientIdentity needed by the implementation to access the system can be defined.
- The implementation requires to support ALECC interface. If not, replace ALECC.subscribe in ACPermission3 instances to other suitable supported methods.

ACClass
METHOD

ACPermission1		**ACPermission2**		**ACPermission3**	
Parameter	Value	Parameter	Value	Parameter	Value
permissionClass	ACClass	permissionClass	ACClass	permissionClass	ACClass
instances	ALE	instances	*	instances	ALECC.subscribe

ACClientIdentity1		**ACClientIdentity2**	
Parameter	Value	Parameter	Value
credentials	*List of ACClientCredentials <Implementation Specific>*	credentials	*List of ACClientCredentials <Implementation Specific>*
roleNames	role1	roleNames	role2

ACRole1		**ACRole2**	
Parameter	Value	Parameter	Value
permissionNames	perm1	permissionNames	perm2

ACPermission4		**ACPermission5**		**ACPermission6**	
Parameter	Value	Parameter	Value	Parameter	Value
permissionClass	ACClass	permissionClass	ACClass	permissionClass	ACClass
instances	ALE.subscribe	instances	ALELR.update	instances	ALETM.defineTMSpec

NOTE 1: The result for invoking any method except getStandardVersion, getVendorVersion, and getSupportedOperations can not be an UnsupportedOperationException. The methods that should raise an UnsupportedOperation exception are those that don't appear in the list of supported operation returned by the getSupportedOperations methods in TCR-T2 step 1.

NOTE 2: Those step that list a specific ALE API name should only be executed if the implementation has implemented the named API.

Step	**Step description**	**Expected results**

1	Invoke the definePermission method to define ACPermission1 (defined in pre-test condition) with permName = “perm1”.	The permission is defined.
2	Invoke getPermissionNames method.	Verify that the list returned includes only “perm1”.
3	Invoke getPermission method with the permName “perm1”.	Verify that ACPermission1 is returned.
4	Invoke defineRole method to define ACRole1 (defined in pre-test condition)with roleName = “role1”.	The role is defined.
5	Invoke getRoleNames method.	Verify that the list returned includes only “role1”.
6	Invoke getRole method with the roleName “role1”.	Verify that ACRole1 is returned.
7	Invoke the updatePermission method with permName “perm1” and ACPermission2.	The permission is updated.
8	Invoke getPermission method with the permName “perm1”.	Verify that ACPermission2 is returned.
9	Invoke the definePermission method to define ACPermission3 (defined in pre-test condition) with permName = “perm2”.	The permission is defined.
10	Invoke the addPermissions method with roleName = “role1” and permissionNames = “perm2”.	The permission is added to “role1”.
11	Invoke getRole method with the roleName “role1”.	Verify that ACRole1 returned contains “perm1” and “perm2”.
12	Invoke the removePermissions method with roleName = “role1” and permissionNames = “perm2”.	The permission is removed from “role1”.
13	Invoke getRole method with the roleName “role1”.	Verify that ACRole1 returned contains “perm1”.
14	Invoke defineRole method to define ACRole2 (defined in pre-test condition)with roleName = “role2”.	The role is defined.
15	Invoke getRoleNames method.	Verify that the list returned includes “role1” and “role2”.
16	Invoke the defineClientIdentity method with identityName “Client1” and ACClientIdentity1 defined in pre-test condition.	The ClientIndentity is defined.
17	Invoke getClientIdentityNames method.	Verify that the list returned includes only “Client1”.
18	Invoke the getClientIdentity method with identityName “Client1”.	Verify that ACClientIdentity1 is returned.
19	Invoke the getClientPermissionNames with identityName “Client1”.	Verify that only “perm1” is returned.
20	Invoke the updateRole method with roleName “role1” and ACRole2.	The role is updated.
21	Invoke getRole method with the roleName “role1”.	Verify that ACRole2 is returned.
22	Invoke the getClientPermissionNames with identityName “Client1”.	Verify that only “perm2” is returned.
23	Invoke the updateRole method with roleName “role1” and ACRole1.	The role is updated.

24	Invoke getRole method with the roleName “role1”.	Verify that ACRole1 is returned.
25	Invoke the addRoles method with identityName “Client1” and “role2”.	The role is added to “Client1”.
26	Invoke the getClientPermissionNames with identityName “Client1”.	Verify that “perm1” and “perm2” are returned.
27	Invoke the getClientIdentity method with identityName “Client1”.	Verify that returned ACClientIdentity1 contains both “role1” and “role2”.
28	Invoke the removeRoles method with identityName “Client1” and “role2”.	The role is removed from “Client1”.
29	Invoke the getClientPermissionNames with identityName “Client1”.	Verify that only “perm1” is returned.
30	Invoke the getClientIdentity method with identityName “Client1”.	Verify that returned ACClientIdentity1 contains only “role1”.
31	(Writing API) “Client1” invokes the define method of the writing API to define a valid CCSpec on the ALECC interface.	Verify that the ALECC does not raise any SecurityException and the CCSpec is defined.
32	Invoke the updateClientIdentity method with identityName “Client1” and ACClientIdentity2 defined in pre-test condition. (This step is always invoked regardless of the API being tested.)	ClientIdentity is updated.
33	(Writing API) Invoke the updatePermission method with permName “perm2” and ACPermission3	The permission is updated.
34	(ALECC only Invoke getPermission method with the permName “perm2”.	Verify that ACPermission3 is returned.
35	(Writing API) Invoke the getClientIdentity method with identityName “Client1”.	Verify that ACClientIdentity2 is returned.
36	(Writing API) Client1 invokes the define method of the writing API to define a valid CCSpec on the ALECC interface.	Verify that the ALECC implementation raises SecurityException.
37	(Reading API) Invoke the updatePermission method with permName “perm2” and ACPermission2	The permission is updated.
38	(Reading API) “Client1” invokes the define method of the readingAPI to define a valid ECSpec on the ALE interface.	Verify that the ALE does not raise any SecurityException and the ECSpec is defined.
39	(Reading API) Invoke the updatePermission method with permName “perm2” and ACPermission4	The permission is updated.
40	(Reading API) Invoke getPermission method with the permName “perm2”.	Verify that ACPermission4 is returned.
41	(Reading API) Invoke the getClientIdentity method with identityName “Client1”.	Verify that ACClientIdentity2 is returned.
42	(Reading API) Client1 invokes the define method of the reading API to define a valid ECSpec on the ALE interface.	Verify that the ALE implementation raises SecurityException.

43	(LR API) Invoke the updatePermission method with permName “perm2” and ACPermission2	The permission is updated.
44	(LR API) “Client1” invokes the define method of the logical reader API to define a valid LogicalReader on the ALECC interface.	Verify that the ALECC does not raise any SecurityException and the LRSpec is defined.
45	(LR API) Invoke the updatePermission method with permName “perm2” and ACPermission5	The permission is updated.
46	(LR API) Invoke getPermission method with the permName “perm2”.	Verify that ACPermission5 is returned.
47	(LR API) Invoke the getClientIdentity method with identityName “Client1”.	Verify that ACClientIdentity2 is returned.
48	(LR API) Client1 invokes the define method of the logical API to define a valid LogicalReader on the ALELR interface.	Verify that the ALELR implementation raises SecurityException.
49	(TM API) Invoke the updatePermission method with permName “perm2” and ACPermission2	The permission is updated.
50	(TM API) “Client1” invokes the defineTMSpec method of the logical reader API to define a valid TMSpec on the ALECC interface.	Verify that the ALECC does not raise any SecurityException and the TMSpec is defined.
51	(TM API) Invoke the updatePermission method with permName “perm2” and ACPermission6	The permission is updated.
52	(TM API) Invoke getPermission method with the permName “perm2”.	Verify that ACPermission6 is returned.
53	(TM API) Invoke the getClientIdentity method with identityName “Client1”.	Verify that ACClientIdentity2 is returned.
54	(TM API) Client1 invokes the defineTMSpec method of the Tag Memory API to define a valid TMSpec on the ALETM interface.	No Exceptions should be raised.
55	Invoke setRoles using identityName “Client1” and a roleNames list that only contains “role1”	
56	Invoke getClientIdentityNames method.	Verify that the list returned includes only “Client1” showing only “role1” in the roleName list.
57	Invoke undefinePermission to remove the permission named “perm1”	
58	Invoke undefinePermission to remove the permission named “perm2”	
59	Invoke getPermissionNames to verify no permissions are defined	Verify the getPermissionNames returns an empty list.
60	Invoke undefineRole to remove the role named “role1”	
61	Invoke undefineRole to remove the role named “role2”	
62	Invoke getRoleNames to verify no roles are defined	Verify the getRoleNames returns an empty list.
62	Invoke undefineClientIdentity to remove the client named “Client1”	

64	Invoke getClientIdentityNames to verify no roles are defined	Verify the getClientIdentityNames returns an empty list.
Post-test Conditions: • All defined permissions, roles and clientidentitites and CCSpec have been undefined.		

17.4 TCR-A4 – Exceptions, Access Control API

Exceptions, Access Control API		
TPId: TCR-A4		
Requirement Purpose: This Test Case confirms that the ALE implementation will raise all exceptions as defined in the ALE specification. This covers exceptions raised due to incorrect parameters passed in ALE API methods and exceptions raised due to missing or invalid parameters in an ACPermission. **Requirements Tested:** GM1, AM1, AM3, AM5, AM7, AM8, AM9		
NOTE: Certain combinations of permission- and role-related methods may raise UnsupportedOperationException instead of performing the expected function.		
Pre-test conditions: • No ACPermission is defined Note: The ACPermission used in this Test Case Requirement should be valid except for the conditions specified in step being performed.		
Step	**Step description**	**Expected results**
1	Invoke the getPermission with an unknown perm name.	Verify that the ALEAC implementation raises a NoSuchPermissionException .
2	Invoke the updatePermission method using an unknown name for the permName string.	Verify that the ALEAC implementation raises a NoSuchPermissionException is raised.
3	Invoke the undefinePermission method with an unknown ACPermission name	Verify that the ALEAC implementation raises a NoSuchPermissionException.
4	Invoke the addPermissions method with an unknown perm name.	Verify that the ALEAC implementation raises a NoSuchPermissionException / NoSuchRoleException.
5	Invoke the setPermissions method with an unknown perm name.	Verify that the ALEAC Implementation raises an NoSuchPermissionException / NoSuchRoleException.
6	Invoke the definePermission method with an unknown permissionClass name.	Verify that the ALEAC implementation raises a PermissionValidationException.
7	Invoke the updatePermission method with an unknown permissionClass name (Note: a valid ACPermission must be defined before this step and undefined after this step is completed.)	Verify that the ALEAC implementation raises a PermissionValidationException.
8	Invoke the definePermission method with an invalid instance string for the specified permission class.	Verify that the ALEAC implementation raises a PermissionValidationException.

9	Invoke the updatePermission method with an invalid instance string for the specified permission class.	Verify that the ALEAC Implementation raises an PermissionValidationException.
10	Invoke the definePermission method with an already existing perm name.	Verify that the ALEAC Implementation raises an DuplicatePermissionException.
11	Invoke the updateRole method using an unknown name for the roleName string.	Verify that the ALEAC Implementation raises a NoSuchRoleException.
12	Invoke the getRole with an unknown roleName.	Verify that the ALEAC Implementation raises a NoSuchRoleException.
13	Invoke the undefineRole with an unknown roleName.	Verify that the ALEAC implementation raises a NoSuchRoleException.
14	Invoke the addPermissions Method with an unknown roleName.	Verify that the ALEAC implementation raises a NoSuchRoleException.
15	Invoke the setPermissions method with an unknown roleName.	Verify that the ALEAC implementation raises a NoSuchRoleException.
16	Invoke the removePermissions method with an unknown roleName.	Verify that the ALEAC implementation raises a NoSuchRoleException.
17	Invoke the addRoles method with an unknown roleName.	Verify that the ALEAC implementation raises a NoSuchRoleException.
18	Invoke the setRoles method with an unknown roleName.	Verify that the ALEAC implementation raises a NoSuchRoleException.
19	Invoke the defineRole method with an invalid ACRole.	Verify that the ALEAC implementation raises a RoleValidationException.
20	Invoke the updateRole method with an invalid ACRole.	Verify that the ALEAC implementation raises a RoleValidationException.
21	Invoke the defineRole method with an already existing ACRole. (Note: a valid ACRole must be defined before this step and undefined after this step is completed.)	Verify that the ALEAC implementation raises a DuplicateRoleException.
22	Invoke the updateClientIdentity method with an unknown ACClientIdentity.	Verify that the ALEAC implementation raises a NoSuchClientIdentityException.
23	Invoke the getClientIdentity method with an unknown identityName.	Verify that the ALEAC implementation raises a NoSuchClientIdentityException.
24	Invoke the getClientPermissionNames method with an unknown identityName.	Verify that the ALEAC implementation raises a NoSuchClientIdentityException.
25	Invoke the undefineClientIdentity method with an unknown identityName.	Verify that the ALEAC implementation raises a NoSuchClientIdentityException.
26	Invoke the addRoles method with an unknown identityName.	Verify that the ALEAC implementation raises a NoSuchClientIdentityException.
27	Invoke the removeRoles method with an unknown identityName.	Verify that the ALEAC implementation raises a NoSuchClientIdentityException.

28	Invoke the defineClientIdentity method with an invalid ACClientIdentity.	Verify that the ALEAC implementation raises a ClientIdentityValidationException.
29	Invoke the updateClientIdentity method with an invalid ACClientIdentity. (Note: a valid ACClientIdentity must be defined before this step and undefined after this step is completed.)	Verify that the ALEAC implementation raises a ClientIdentityValidationException.
30	Invoke the defineClientIdentity method with an already existing ACClientIdentity.	Verify that the ALEAC implementation raises an DuplicateClientIdentityException.

651

652

17.5 TCR-A5 – XML Vendor Extension Validaion

XML Vendor Extension Validation		
TPId: TCR-A5		
Requirement Purpose: This Test Case confirms that vendor extensions to the ACPermission, ACRole and ACClientIdentity have been added in accordance with the rules set forth in the ALE 1.1 specification. This TCR is opational. This TCR only needs to be executed for implementation that have vendor extension. **Requirements Tested:** XM1, XM2, XM3, XM4, XM5, XM6, XM8, XM9, XM12		
Pre-test conditions: • The vendor has submitted XML files containing instances of ACPermission, ACRole and ACClientIdentity that contain the the vendor extensions or the vendor's XSD for the Access Control API or appropriate documentation confirming the vendor is the owner of the namespace used for the vendor extensions. • The vendor has provided its XSD files so they can be inspected to ensure that elements, attributes and extensions have not be added in places not allowed by the specification.		
Step	**Step description**	**Expected results**
1	Examine the XML documents, XSD documents or other documentation submitted by the vendor to verify the vendor is the owning authority for the name space used for all vendor attribute and element extensions.	Confirm (by design)
2	Validate the XML ACPermission, ACRole and ACClientIdentity instance documents received in TCR-A1 through A4 against the ALE 1.1 Access Control API XSD. (See section 13.4)	The XML documents should validate successfully.
3	Inspect the vendor XSDs to ensure that elements, attributes and extensions have not be added in places not allowed by the ALE 1.1 specification.	There are no elements, attributes and extensions added in the vendor's XSDs in places not allowed by the ALE 1.1 specificatoin.

653

654

655

18 ALE Logical Reader API

18.1 TCR-L1 – Get Version, Logical Reader API

Get Version, Logical Reader API		
TPId: TCR-L1		
Test Purpose: This Test Case confirms the proper functions of the ALE methods of the Logical Reader API that return the ALE standard version and the vendor version for the ALE implementation under test. The return of correct version numbers also confirms the correct implementation is being tested. **Requirements Tested :** GM1, GM2, GM3, GM4, GM5, LM1, LM10		
Pre-test conditions: • None		
Step	**Step description**	**Expected results**
1	Invoke the getStandardVersion method of Logical Reader API.	• Confirm the string "1.1" is returned. • Confirm the result returned by this method only pertain to the API to the Logical Reader API.
2	Invoke the getVendorVersion method of Logical Reader API.	• Confirm that either an empty string or a string conforming to a proper URI is returned. • Confirm the vendor is the owning authority of the URI if the returned string is not empty (by Design) • Confirm the result returned by this method only pertain to the API to the Logical Reader API.

18.2 TCR-L2 – Defining, Un-defining, Updating, Retrieving LRSpecs, Logical Reader API

Defining, Un-defining, Updating, Retrieving LRSpecs, Logical Reader API		
TPId: TCR-L2		
Test Purpose: This Test Case confirms that a valid Logical Reader can be defined, updated and undefined. Further the defining and un-defining of the Logical Reader can be verified with "ALELR" API methods getLRSpec and getLogicalReaderNames. **Requirements Tested:** GM1, LM1, LM2, LM5, LM10, LM11		
Pre-test conditions: • One or more logical readers (including base readers) are defined.		
Step	**Step description**	**Expected results**
1	Invoke the define method with a valid LRSpec with isComposite = true.	The ALELR implementation contains the LRSpec definition supplied in the define method. Steps 2 and 3 confirm the defining of the LRSpec.
2	Verify the LRSpec was defined by invoking the getLogicalReaderNames method	Verify that the name returned in the list is that of the LRSpec just defined. Also, all base and logical readers in the pre-test conditions visible to the user should be in the list.

3	Invoke getLRSpec using the name of the defined LRSpec.	Verify that the LRSpec returned is the same as the one defined
4	Invoke the define method with a valid LRSpec with isComposite = true	The ALELR implementation contains the LRSpec definition supplied in the define method.
5	Verify the LRSpecs were defined by invoking the getLogicalReaderNames method	Verify that the names returned in the list are that of the LRSpecs defined in step 1 and 4
6	Invoke ndefined to remove the LRSpec that was defined in step 1.	The ALELR implementation should no longer have the LRSpec defined.
7.	Verify that the LRSpec is undefined by invoking the getLogicalReaderNames method.	Verify that the list returned only contain the spec name defined in step 4
8.	Invoke the update method with a valid and different LRSpec to the reader defined in step 4	The ALELR implementation will update the LRSpec definition of the corresponding reader.
9	Invoke getLRSpec using the name of the defined LRSpec in step 4	Verify that the LRSpec returned is the same as the one updated
10	Invoke ndefined to remove the LRSpec that was defined in step 4.	The ALELR implementation should no longer have the LRSpec defined.
11	Verify that the LRSpec is undefined by invoking the getLogicalReaderNames method.	Verify that the list does not contain the logical reader that was defined in step 4 and undefined in step 10. The logical readers defined in the pre-test conditions should still be returned.

662

663 18.3 TCR-L3 – Adding, Setting, Removing Readers, Logical
664 Reader API

Adding, Setting, Removing Readers, Logical Reader API		
TPId: TCR-L3		
Test Purpose: This Test Case confirms that a valid Logical Reader can be added, set and removed. **Requirements Tested:** GM1, LM1, LM5, LM10		
Pre-test conditions: • One or more logical readers (including base readers) are defined.		
Step	**Step description**	**Expected results**
1	Invoke the define method with a valid LRSpec with isComposite = true.	The ALELR implementation contains the LRSpec definition supplied in the define method.
2	Invoke getLRSpec using the name of the defined LRSpec.	Verify that the LRSpec returned is the same as the one defined
3	Add a new reader that is not in defined LR using addReaders	New reader will be added in the LRSpec
4	Invoke getLRSpec using the name of the defined LRSpec.	Added reader will be seen in the returned LRSpec
5	remove the added reader in step 3 using removeReaders	Removed reader will not be seen in the returned LRSpec

6	Set a new list of readers in the current LRSpec invoking setReaders method	Reader list in the corresponding LRSpec will be set new reader list
7	Invoke getLRSpec using the name of the defined LRSpec.	New reader list will be seen in the returned LRSpec
8	Invoke undefined to remove the LRSpec that was defined in step 1.	The ALELR implementation should no longer have the LRSpec defined.

665

666 18.4 TCR-L4 – Tag Smoothing – Setting and Retrieving Relevant 667 Properties of a Reader, Logical Reader API

Tag Smoothing – Setting and Retrieving Relevant Properties of a Reader, Logical Reader API

TPId: TCR-L4

Test Purpose: This Test Case confirms that a valid Logical Reader's property can be get and set.
Requirements Tested: GM1, LM1, LM3, LM5, LM7, LM9, LM10, LM11, LM14, LM15, LM16, LM17

NOTE: If the ALE 1.1 implementation under test indicates that it does not support Tag Smoothing, it may raise appropriate ValidationExceptions.

Pre-test conditions:

- One or more logical readers (including base readers) are defined.
- The ALELR implementation shall support "Tag Smoothing" and all of its parameters.
- A valid ECSpec is defined as follows:

ECSpec					
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
Reader List	1 Reader			startTrigger	Null
stopTrigger	Null	startTriggerList	Null	stopTriggerList	Null
duration	5 Sec	stableSetInterval	0	repeatPeriod	8 sec
Current	Yes	Additions	No	Deletions	No
includeCount	No	includeEPC	Yes	includeTag	No
reportIfEmpty	False	reportOnlyOnChange	False	includeSpecInReports	False
includeRawHex	No	includeRawDecimal	No	groupSpec	No
includePatterns	No	excludePatterns	No	primaryKeyFields	Null
filterList	No	statProfileNames	No		

Step	Step description	Expected results
1	Invoke the define method with a valid LRSpec, name = "LR1", isComposite = false.	The ALELR implementation contains the LRSpec definition supplied in the define method.
2	Invoke the setProperties method using the following LRProperty values for the logical reader defined in step 1:- GlimpsedTimeout = 500 ms; ObservedTimeThreshold = 3000 ms; LostTimeout = 2000 ms.	All properties will be set.

3	Invoke the getPropertyValue method using the property name GlimpsedTimeout.	Verify that 500 ms value will be returned.
4	Invoke the getPropertyValue method using the property name ObservedTimeThreshold.	Verify that 3000 ms value will be returned.
5	Invoke the getPropertyValue method using the property name LostTimeout.	Verify that 2000 ms value will be returned.
6	Invoke the getPropertyValue method using the property name ObservedCountThreshold.	Verify that ALELR implementation returns an empty string.
7	Invoke the subscribe method using the ECSpec defined in pre-test condition and begin the event cycle.	The subscription is successful.
8	Immediately place a tag in the reader field (for a period greater than 3500 ms for the tag being in 'Observed' state).	Verify that after 5 seconds an ECReports that conforms to the ALE XSD should be returned containing the tag in the reader field.
9	Remove the tag from the reader field immediately after receciving the ECReports in step 8 (For the tag being in 'Unknown' state at the beginning of next event cycle).	Verify that after 5 seconds of the beginning of the next event cycle, no ECReports is received.
10	Invoke the unsubscribe method to unsubscribe the ECSpec in step 7.	The unsubscription is successful.
11	Invoke the setProperties method using the following LRProperty value for the logical reader defined in step 1:- LostTimeout = 5000 ms.	The property value is reset.
12	Invoke the subscribe method using the ECSpec defined in pre-test condition and begin the event cycle.	The subscription is successful.
13	Immediately place a tag in the reader field (for a period greater than 3500 ms for the tag being in 'Observed' state).	Verify that after 5 seconds an ECReports that conforms to the ALE XSD should be returned containing the tag in the reader field.
14	Remove the tag from the reader field immediately after receciving the ECReports in step 13 (For the tag being still in 'Observed' state at the beginning of next event cycle).	Verify that after 5 seconds an ECReports that conforms to the ALE XSD should be returned containing the tag in the reader field.
15	Invoke the unsubscribe method to unsubscribe the ECSpec in step 12.	The unsubscription is successful.
16a.	Undefine the ECSpec.	The ECSpec is undefined.
16b.	Invoke undefine to remove the LRSpec that was defined in step 1.	The ALELR implementation should no longer have the LRSpec defined.

668

669 18.5 TCR-L5 – Exceptions, Logical Reader API

Exceptions, Logical Reader API
TPId: TCR-L5

Test Purpose: This Test Case confirms that ALELR implementation will raise all exceptions as defined in the ALELR specification.
Requirements Tested: GM1, LM1, LM4, LM4, LM5, LM6, LM9, LM10, LM12, LM13, LM18

Pre-test conditions:

- One or more Logical Readers (including base readers) are defined

Step	Step description	Expected results
1	Invoke the define method with a valid LRSpec isComposite = true and name = "LR1".	The ALELR implementation contains the LRSpec definition supplied in the define method.
2	Invoke the define method with a valid LRSpec isComposite = true and name = "LR1".	Verify that the ALELR implementation raises a DuplicateNameException.
3	Invoke the define method with a valid LRSpec isComposite = false and name = "LR2" and GlimpsedTimeout = -1, ObservedTimeThreshold = -1, ObservedCountThreshold = -1, LostTimeout = -1	Verify that the ALELR implementation raises a ValidationException
4	Invoke the update method with a valid LRSpec and name = "LR3".	Verify that the ALELR implementation raises a NoSuchNameException
5	Invoke the update method with an invalid LRSpec and name = "LR1".	Verify that the ALELR implementation raises a ValidationException
6	Invoke the define method with a valid LRSpec isComposite = true and name = "LR2".	The ALELR implementation contains the LRSpec definition supplied in the define method.
7	Define a ECSpec using LR2 as logical reader duration = 5 sec repeatPeriod = 10 sec stablesetInterval = 0, reportIfEmpty = false.	The ALELR implementation contains the ECSpec definition supplied in the define method.
8.	Invoke the subscribe method to activate the ECSpec and begin the event cycle	Subscribe returns void
9	Wait 2 sec.	
10	Invoke the update method with a valid LRSpec and name = "LR2".	Verify that the ALELR implementation raises an InUseException or provide documentation for what "as soon as possible" means and provides documentation.
11.	Invoke unsubscribe method.	UnSubscribe returns void
12.	Invoke the update method with a valid LRSpec and name = < externally-defined reader name>	Verify that the ALELR implementation raises a ImmutableReaderException
13.	Invoke the update method with a valid LRSpec which contain "LR1" reader, and name = "LR1".	Verify that the ALELR implementation raises a ReaderLoopException
14.	Repeat step 4 (no LRSpec needed) and steps 8-12 (no LRSpec needed in any step) for undefine method.	
15.	Invoke the getLRSpec method with name = "LR3".	Verify that the ALELR implementation raises a NoSuchNameException
16.	Invoke the addReaders method with name = "LR3".	Verify that the ALELR implementation raises a NoSuchNameException

17.	Invoke the addReaders method with name = "LR1" and the reader list will contain an unknown reader.	Verify that the ALELR implementation raises a ValidationException
18.	Repeat steps 8-11 (no LRSpec needed in any step) for addReaders method.	Verify that the ALELR implementation raises an InUseException provide documentation for what "as soon as possible" means and provides documentation.
19.	Invoke the addReaders method with a valid name = "< externally-defined reader name>	Verify that the ALELR implementation raises a ImmutableReaderException (or a NonCompositeReaderException if it is a *base* reader).
20.	Invoke the define method with a valid LRSpec isComposite = false and name = "LR4".	The implementation should raise a ValidationException unless API defined base readers are supported. Then, the ALELR implementation contains the LRSpec definition supplied in the define method.
21.	Invoke the addReaders method with name = "LR4" and the reader list will contain all known readers.	The implementation should raise a NoSuchNameException unless API defined base readers are supported. Then, verify that the ALELR implementation raises a NonCompositeReaderException
22.	Invoke the addReaders method with a valid list of known readers which contain "LR1" reader, and name = "LR1".	Verify that the ALELR implementation raises a ReaderLoopException
23.	Repeat steps 16-22 for setReaders method (no LRSpec needed in any step).	Note: Step 20 should produce a DuplicateNameException if API-defined base readers are supported
24.	Repeat steps 16 and 18-21 for removeReaders method (no LRSpec needed in any step).	Note: Step 20 should produce a DuplicateNameException if API-defined base readers are supported.
25.	Invoke the setProperties method with valid LRProperty list and name = "LR3".	Verify that the ALELR implementation raises a NoSuchNameException
26.	Invoke the setProperties method with invalid LRProperty list and name = "LR1".	Verify that the ALELR implementation raises a ValidationException
27.	Repeat step 8-11 for setProperties method.	Verify that the ALELR implementation raises an InUseException provide documentation for what "as soon as possible" means and provides documentation.
28.	Invoke the setProperties method with name = contain < externally-defined reader name>.	Verify that the ALELR implementation raises a ImmutableReaderException
29	Undefine "LR2"	Verify that the ALELR implementation raises an InUseException
30.	Undefine the defined ECSpec.	
31.	Undefine "LR1", "LR2", "LR4".	Note: Step 20 should produce a NoSuchNameException if API-defined base readers are not supported."

670

671 ## 18.6 TCR-L6 – Using Composite, Logical Reader API

672

Using Composite, Logical Reader API

TPId: TCR-L6

Test Purpose: This Test Case confirms that a composite logical reader can be used by an ECSpec / CCSpec.
Requirements Tested: GM1, LM1, LM12

Pre-test conditions:

- No Logical Reader is defined.

LRSpec	
isComposite	true
readers	*<list of valid logical readers>*
properties	Null

Step	Step description	Expected results
1	Invoke the define method with LRSpec defined in pre-test condition and name = "LR1".	The ALELR implementation contains the LRSpec definition supplied in the define method.
2	Invoke getLRSpec using "LR1" as the name of the logical reader.	Verify that the LRSpec returned is the same as the one defined in step 1.
3.	Invoke a define method with a valid ECSpec using logicalReader = "LR1".	Verify that the ALE implementation accepts the ECSpec definition.
4.	Repeat step 3 for a CCSpec.	

673

674 18.7 TCR-L7 – XML Vendor Extension Validaion

XML Vendor Extension Validation		
TPId: TCR-L7		
Requirement Purpose: This Test Case confirms that vendor extensions to the LRSpec have been added in accordance with the rules set forth in the ALE 1.1 specification. This TCR is opational. This TCR only needs to be executed for implementation that have vendor extension. **Requirements Tested:** XM1, XM2, XM3, XM4, XM5, XM6, XM8, XM9, XM12		
Pre-test conditions: • The vendor has submitted XML files containing instances of LRSpec that contain the the vendor extensions or the vendor's XSD for the Logical Reader API or appropriate documentation confirming the vendor is the owner of the namespace used for the vendor extensions. • The vendor has provided its XSD files so they can be inspected to ensure that elements, attributes and extensions have not be added in places not allowed by the specification.		
Step	**Step description**	**Expected results**
1	Examine the XML documents, XSD documents or other documentation submitted by the vendor to verify the vendor is the owning authority for the name space used for all vendor attribute and element extensions.	Confirm (by design)

2	Validate the XML LRSpec instance documents received in TCR-l1 through L6 against the ALE 1.1 Logical Memory API XSD. (See section 13.4)	The XML documents should validate successfully.
3	Inspect the vendor XSDs to ensure that elements, attributes and extensions have not be added in places not allowed by the ALE 1.1 specification.	There are no elements, attributes and extensions added in the vendor's XSDs in places not allowed by the ALE 1.1 specificatoin.

675
676
677

19 References

679

680 [ALE1.1 Part1] EPCglobal, "The Application Level Events (ALE) Specification,
681 Version 1.1 Part I: Core Specification," EPCglobal Proposed Specification V, xx
682 December 2007.
683
684 [ALE1.1 Part2] EPCglobal, "The Application Level Events (ALE) Specification,
685 Version 1.1 Part II: XML and SOAP Bindings," EPCglobal Proposed Specification, xx
686 December 2007.
687
688 [Unicode] The Unicode Consortium, The Unicode Standard, Version 5.0, Addison-
689 Wesley, November, 2006, ISBN 0321480910.

20 Acknowledgement of Contributors and of Companies Opt'd-in during the Creation of this Standard (non-normative)

693 *Disclaimer*
694 *Whilst every effort has been made to ensure that this document and the information*
695 *contained herein are correct, EPCglobal and any other party involved in the creation of*
696 *the document hereby state that the document is provided on an "as is" basis without*
697 *warranty, either expressed or implied, including but not limited to any warranty that the*
698 *use of the information herein with not infringe any rights, of accuracy or fitness for*
699 *purpose, and hereby disclaim any liability, direct or indirect, for damages or loss*
700 *relating to the use of the document.*
701
702 Below is a list of active participants and contributors in the development of the ALE 1.1
703 specification. This list does not acknowledge those who only monitored the process
704 without contributing or those who chose not to have their name listed here. An "active
705 participant" for the purpose of this list is an individual who corresponded using the
706 Working Group mailing list or who attended one or more face-to-face or teleconference
707 meetings of the Working Group.
708 Mark Frey (EPCglobal Inc.), Facilitator
709 Richard Bach (GlobeRanger), Co-Chair, Conformance Requirements Editor

Andreas Kerschbaumer (EB [formerly 7iD])
Soumya Roy Chowdhury (Polaris Networks), Drafted Content
Ken Traub (Ken Traub Consulting LLC), Co-Chair
Arun Badami (MET Laboratories)
Bud Biswas (Polaris Networks)
Ted Osinski (MET Laboratories)
Wolfgang Thaller (EB[formerly 7iD])
John Ross (IBM)
Hemant Sahgal (Iris Software)

The following list enumerates, in alphabetical order by company name, all companies that signed the EPCglobal IP Policy and the opt-in agreement for the EPCglobal Working Group that created the ALE 1.1 standard.

7iD Technologies (formerly EOSS GmbH)
Accenture
Acer Cybercenter Service Inc.
ACSIS
Afilias Limited
Allixon Co., Ltd
Altria Group, Inc./Kraft Foods
Alvin Systems
AMCO TEC International Inc.
AMOS Technologies Inc.
Applied Wireless (AWiD)
Auto-ID Labs - Cambridge
Auto-ID Labs - ICU
Auto-ID Labs - Japan
Auto-ID Labs - MIT
BEA Systems
Cheng-Loong Corporation
Cisco
City Univ of Hong Kong
Cognizant Technology Solutions
Convergence Sys Ltd
Dai Nippon Printing (DNP)
Denso Wave Inc
Elektrobit (formerly 7iD)
ECO, Inc.
EPCglobal Inc.
ETRI - Electronics and Telecommunication Research Institute
FEIG Electronics
France Telecom
Fujitsu Ltd
GlobeRanger

756	GS1 Australia EAN
757	GS1 Germany (CCG)
758	GS1 Hong Kong
759	GS1 International
760	GS1 Japan
761	GS1 Netherlands (EAN.nl)
762	GS1 South Korea
763	GS1 Sweden AB (EAN)
764	GS1 Taiwan (EAN)
765	GS1 UK
766	GS1 US
767	Hewlett-Packard Co. (HP)
768	IBM
769	Impinj
770	Institute for Information Industry
771	Intermec
772	Iris Software
773	Ken Traub Consulting LLC
774	Kimberly-Clark
775	KL-NET
776	KTNET - Korea Trade Network
777	Leiner Health Products Inc.
778	LG CNS
779	Research Center for Logistics Information Technology (LIT)
780	Lockheed Martin - Savi Technology Divison
781	Manhattan Associates
782	MET Laboratories
783	MetaBiz
784	MetaRights, Ltd.
785	Metro
786	Microelectronics Technology, Inc.
787	Mstar Semiconductor
788	NEC Corporation
789	Nippon Telegraph & Telephone Corp (NTT)
790	noFilis Ltd.
791	Nomura Research Institute
792	NXP Semiconductors
793	NYSSA S.R.L.
794	OatSystems
795	Oracle Corporation
796	Panda Logistics Co.Ltd
797	Pango Networks, Inc.
798	Polaris Networks
799	Polaris Networks
800	Printronix
801	Psion Teklogix Inc.

Q.E.D. Systems
Rafcore Systems Inc.
Red Prairie
Regal Scan Tech
RetailTech
Reva Systems
RF-IT Solutions GmbH
RFID Research Center, Chang Jung Christian University
rfXcel Corp
Samsung SDS
Sandlinks
SAP Aktiengesellschaft
Secure RF
Sedna Systems, Ltd.
Shipcom Wireless, Inc.
Sirit Technologies Inc
Sirit Technologies Inc
Supply Insight, Inc.
SupplyScape Corporation
Tagent Corporation
The Boeing Company
ThingMagic, LLC
Tibco Software, Inc
Toppan Printing Co., Ltd
Toray International, Inc.
Tracetracker Inovation AS
TrueDemand Software
Userstar Information System Co. Ltd
Ussen Limited Company
VeriSign
Vue Technology
Wal-Mart Stores, Inc.
Waldemar Winckel GmbH & Co. KG
Warelite Ltd

