Errata in "Information Management – Extensible Access Method (XAM) – Part 1: Architecture" v1.0

SNIA FCAS TWG, June 16, 2009 – Approved Version

Chapter 3: Summary of errata:

- Add normative definitions of fatal error, non-fatal error, and corrupt.

Chapter 6: Summary of errata:

- Explain what an XStream instance is when the concept is introduced
- Clarify that a language binding may use classes for the XUID and date stypes.
- Specify that the CRC field of a XUID is in network byte order.
- Clarify wording of specification text for XUID fields.
- Add explanation of what XStreams are used for.
- Restrict XStream to be openable in one mode per instance of primary object
- Clarify several uses of "XStream" that should be "XStream instance"
- Explain that end of stream indication is language-specific.
- Explain that any XStream.read operation may read zero bytes.

Chapter 7: Summary of errata:

- Fix minor format problem in Table 16.
- Add additional logging fields from the XAM SDK

Chapter 8: Summary of errata:

- Add statement that XSets are also used to run jobs.
- Clarify that XSet.commit is all-or-nothing atomic and subject to communication failures
- Clarify when commit time is updated - only on commit
- Clarify distributed access example.
- Clarify text that describes policy properties and policy lists
- XSet Export Example: Use correct MIME type for XStream, and clarify the boolean value
- Clarify that halting a completed asynchronous operation is not an error.
- Correct policy handling specification for import and clarify what happens when an XSet with the same XUID already exists.
- Consistently use correct field names for query job fields.
- Clarify that job .errorhealth field is only created when an error occurs.
- Correct job field names to use ".xam" prefix.
- Clarify system field export behavior
- Export format: Correct specification of policy w/example, clarify what TOC offset points to.
- Correct state name in XAsync FSM table (Table 54)

Chapter 9: Summary of errata:

- Add error case to XSet.createRetention, as "event" retention is always binding.
- Allow retention duration to be zero.
- Consistently use correct field names for autodelete and shred fields.
- Consistently use <name> token to stand for a policy name

Chapter 10: Summary of errata:

- Use a valid field name in first query example.
- Add type comparison validity rules for booleans.
- Correct operator error in query example in footnote.
- Clarify the distance parameter to the Level 2 *within* operator.
- Explain what happens when a field does not exist - exists() works, nothing else does.
- Allow a boolean property in a query on its own without an operator.
- Correct query rule typo that incorrectly allows binary operators to be omitted.
- Fix several XAM QL grammar problems that allowed invalid numeric literals.
- Use appropriate quote characters in the XAM QL grammar.
- Add specification of whitespace usage in query language.
- Add "date" and "xuid" to list of reserved key words.
- Query job-related fields are read-only only while the query job is running.
- Consistently use correct field names for query job fields.
- Explain that the query job results XStream contains all the results, and hence end of stream cannot be indicated while the corresponding query job is running.
- Clarify that halting a query job is not an error; hence error fields are not created.
- Clarify that job health field may not exist.
- Use correct names of authorization granules that control jobs
- Add an example of mixed type query

Annex B: Summary of errata:

- Use correct XML double type in canonical XSet interchange format instead of float.
- Add missing XML specification of XUID type

3.1 Terms: Errors

Insert the following three terms, renumbering other definitions as needed for alphabetical order:

- **corrupt**
  The state of an object instance that results from a fatal error. The only method allowed on a corrupt object instance is to abandon it; all other invoked methods cause errors.

- **fatal error**
  A method result indicating that the invoked method failed to completely perform its intended operation. A fatal error indicates that the object instance on which the method was invoked has transitioned to the corrupt state.

- **non-fatal error**
  A method result indicating that the invoked method did not perform its intended operation. A non-fatal error shall not cause an object instance to change its current state.

6.2 XAM Secondary Objects: XStream instance explanation:

The following sentence in the first paragraph does not explain what an XStream instance is:

> XStreams are a type of field, and XStream instances can be used to store and retrieve data.

Replace it with the following sentence:

> An XStream is a field of the primary object to which the XStream is attached. It is necessary to open an XStream in order to store and retrieve the data that it contains; an opened XStream is called an XStream instance. Opening an XStream creates an XStream instance within the primary object instance and returns an XStream handle that is used to store and retrieve data.

6.3.3 Properties: XUIDs and dates may be classes

Add the following sentence as its own paragraph after the current Note following Table 5:

> A language binding of XAM may use XAM-specific classes or similar constructs for the xuid and xam_datetime stypes rather than representing them as byte strings or character strings. The methods for any such class are specified in the XAM language binding specification for that language.

6.3.4 XUID Format: CRC and wording clarifications

Make the following changes to the bulleted list following Figure 5 (XUID Format) to state requirements and specify the format of the CRC field:

- 2nd bullet - Change "is the SNMP enterprise number" to "shall be the SNMP enterprise number".
- 4th bullet - Change "contains the full length" to "shall contain the full length.
- 5th bullet - Change "The CRC field allows" to "The CRC field shall contain a 2-byte (16-bit) CRC in network byte order. The CRC field enables"

6.5.1 Operating on XStreams: XStream usage

Add the following two paragraphs to the beginning of Section 6.5.1 to explain XStream usage:

> XStreams are used for two major purposes in XAM:
>
> 1. Storage and retrieval of data - the XStream stores data for a field, usually an XSet field.
> 2. Import and export - a special XStream is created to import or export the contents of an XSet (see Section 8.8, "XSet Import and Export).
>
> When an XStream stores data for a field, the field consists of the XStream plus the field attributes (see Section 6.3.2, "Field Attributes"). For example, an XStream that stores data for a field has no subtype beyond XStream because the field type is contained in the Type attribute of the field.
>
> An important XStream-based field is the field that contains query results (see Section 10.7.1, "Query Job Specific XSet Fields"); the end of this XStream cannot be determined until all the query results are produced, hence reading this XStream while the corresponding query job is active may result in no bytes being read without indicating that the end of the XStream has been reached.

6.5.1 Operating on XStreams: Open mode restrictions

Add the following paragraph to Section 6.5.1 after the appendonly bullet item (this paragraph is not bulleted) to state restrictions on XStream open modes:

> Within a single instance of the parent primary object that contains the XStream, the XStream shall not be open in more than one mode at any point in time. Within a single primary object instance, an XStream may have multiple open instances simultaneously in readonly mode, but shall not have more than one open instance at any time in writeonly mode or appendonly mode. Attempting to open an XStream with a mode that would violate these restrictions shall cause a non-fatal error.

6.5.1 Operating on XStreams: XStream instance clarification

The last paragraph before Table 10 begins as follows:

> Once operations on the XStream are complete, the XAM application is expected to close the XStream, which causes all resources associated with the XStream instance to be freed. Note that any changes made to an XStream are only associated with the instance of the parent object (i.e., XAM Library, XSystem, or XSet).

Every occurrence of "XStream" in the above two sentences should be "XStream instance". Change the first, second and fourth occurrences of "XStream" to "XStream instance" to accomplish this.

Also, in Table 10, add the word "instance" to the Description of XStream.close, so that the cell contains:

> Close the XStream instance.

6.5.1 Operating on XStreams: End of Stream indicator

In Table 10 (XStream Methods), add the following sentence to the Description of XStream.read:

> Indicate when the end of the XStream is reached.

Add the following sentence to the Note that follows the table:

> The indication that XStream.read has reached the end of the XStream is specific to the language binding.

6.5.1 Operating on XStreams: Reading zero bytes

Add the following two paragraphs to the end of the section after the Note that follows Table 10:

> Any invocation of XStream.read on any XStream may result in zero bytes being read without indicating that the end of the XStream has been reached. XAM implementations should do this only when a significant delay is expected before more bytes become available for reading from the XStream (e.g., this may be appropriate for the results XStream of a query job that takes a significant amount of time to execute; the end of XStream indication cannot be returned until the job is complete).
>
> Immediately retrying an XStream.read that reads zero bytes is not recommended; instead, the XAM application should wait for some period of time before retrying or should use the asynchronous version of XStream.read (see Section 8.10 "Asynchronous Operations").

6.6 FSMs for Secondary Objects – XStreams and XIterators: Clarify instance

In the first sentence of this section, change "the XStream and XIterator instance." to "the XStream instance and the XIterator instance."

7.1.4 Fields of the XAM Library Object: Table 16

In the Field Name column, the field name ".xam.log.path" is not italicized. Italicize it to match the other entries in that column.

7.1.4 Fields of the XAM Library Object: Additional Logging Fields

Add the following fields to Table 16:

Field Name	Type	Binding	ReadOnly
.xam.log.append	xam_boolean	FALSE	FALSE
.xam.log.max.rollovers	xam_int	FALSE	FALSE
.xam.log.max.size	xam_int	FALSE	FALSE

Add the following descriptions of these fields:

.xam.log.append

is a xam_boolean value indicating whether to append to an existing log file (TRUE) or overwrite (FALSE). The default value shall be FALSE.

.xam.log.max.rollovers

is a xam_int value indicating the number of previous log files to retain when starting a new log file. The default value shall be 1.

.xam.log.max.size

is a xam_int value indicating the maximum size in bytes that a log file may reach before a new log file is started. The default value shall be 1GB (2^30 = 1,073,741,824 bytes).

8.1 XSet Behavior: Jobs

Add the following sentence to the end of the first paragraph of Section 8.1:

XSets can also be used to run and control jobs (See Section 8.9 " XAM Jobs and XAM Job Control"), including the query job (See Chapter 10 "Query").

8.1 XSet Behavior: Commit is atomic and subject to communication failures

Add the following sentences to the end of the third paragraph of Section 8.1:

XSet.commit shall be an atomic operation. If an invocation of XSet.commit cannot persist all of the changes made to the XSet, the invocation shall preserve the unmodified contents of the XSet, shall not generate a new XSet, and shall cause an error (fatal or non-fatal).

Add the following sentences to the end of the fourth paragraph of Section 8.1:

Note that communication failures or errors may create situations in which XSet.commit completes successfully at the XAM Storage System, but that success is not communicated to the XAM Application. In such a situation, the XSet may be successfully committed, but the XSet.commit invocation may return a fatal or non-fatal error.

8.2.3 Normative XSet Fields: Clarify commit time update

The last sentence in the second bullet for *.xset.time.commit* is:

> Opening an XStream for write shall cause *.xset.time.commit* to be updated.

Change "updated" to "updated when the XSet is committed" for clarity.

8.6.2.2 XSet Conflict Resolution, Example 2: Clarify corruption cause

The last sentence in the description of this example is:

> Note that process A is still vulnerable to process B causing changes to the XSet, which would result in process A's XSet instance transitioning to the corrupt state when process A tries to read XSet fields.

This sentence is correct from the point of view of process A, as that process has to be prepared for the XSet instance to become corrupt, but the sentence incorrectly implies that any attempt to read fields causes corruption. Replace that sentence with the following rewritten sentence:

> Note that process A is still vulnerable to process B causing changes to the XSet, which may result in process A's XSet instance transitioning to the corrupt state when process A tries to read XSet fields that the VIM has to obtain from the XAM Storage System.

8.7 XSet Policy: Policy property and property list clarification

The second paragraph in Section 8.7 is currently:

> The XAM specification of an XSet policy is an XSet system property with XSet policy methods for creating, updating, and removing an XSet policy property. An XSet policy property value is referred to as the policy name. The XAM specification of an XSystem policy list is a set of XSystem properties, where each element of the set is an XSystem property with a policy name value that is identical to the policy name that is appended at the end of the XSystem property name. Unless otherwise specified, the XSystem policy list may be the empty set, i.e., no XSystem policy list properties exist. The attributes of the XSet policy and XSystem policy list properties are specified in Table 48.

This text has been found to be unclear by reviewers and implementers. For clarity, replace the above paragraph with the following two paragraphs:

> An XSet system property is used to store the applicable policy for each area of agreement; this property is called an XSet policy property. XSet policy properties shall have a readonly attribute of TRUE. For each XSet policy property, a corresponding method is specified to apply policy in that area to the XSet; the method shall apply a named policy (provided as a parameter to the method), shall set the value of the corresponding XSet policy property to the policy name, and shall create the XSet policy property if it does not exist. Reset methods are specifies to remove XSet policy properties, and a single reset method may remove multiple XSet policy properties. For XSet policy properties, the apply and reset methods take the place of the general field create, set and delete operations specified in Section 6.4 "Methods that Operate on Fields" for XSet policy properties because a XAM Application cannot use the general field methods to create system fields, set readonly fields, or delete readonly fields.
>
> The set of policy names that may be used as the value of an XSet policy property are stored in a corresponding policy list in the XSystem that contains the XSet. An XSystem policy list is a set of XSystem properties whose names have a common prefix that designates the area of (policy) agreement to which the policies apply. A property in an XSystem policy list shall contain the policy name value that may be used in the corresponding XSet policy property, and that value shall be the final name component of the XSystem property name (i.e., the XSystem property name is required to end with *.<policy name>* where *<policy name>* is the value of the property). Unless otherwise specified, an XSystem policy list may be empty (i.e., no XSystem policy list properties exist for a specified common name prefix) . The attributes of the XSet policy and XSystem policy list properties are specified in Table 48.

In addition, in Table 48, change the Readonly entry for *.xset.<XAM standard name>.policy* from "XAM specified" to "TRUE".

8.7 XSet Policy: Figure 17

The text starting with the number "2" is:

> 2. The XSystem must honor the "bar val" policy for the othe XSet.

Replace "othe" with "other" to correct the typo.

8.8.1 XSet Export Process: XSet terminology

The second sentence in item 4 under "Exporting an XSet" is:

> This close shall return the XSet to the state it was in before the export operation.

Change "XSet" to the correct "XSet instance" in that sentence.

8.8.1 XSet Export Process: System field export

Change the second bullet item after the second paragraph from:

> Shall export all XAM standard system fields

to:

> Shall export all XAM system fields

8.8.2 XSet Import Process: Correct system field reference

On p.90, in the second bullet, change "XAM standard system fields" to "XAM system fields".

8.8.2 XSet Import Process: XSet with same XUID exists

On p.90, the last bullet on the page (continues onto p.91) is:

> If an XSet with the same XUID exists in the importing XAM Storage System, the existing XSet shall be overwritten with the imported XSet on commit of the imported XSet. An XSystem may generate a fatal or non-fatal error on commit, if a field update would result in a violation of specialized field rules (e.g., cause a retention field to decrease), or if a binding policy is part of the XSet and cannot be enforced on the importing XSet.

The check is at XSystem scope, not XAM Storage System scope, and the possible commit errors need clarification. Change this to:

> If an XSet with the same XUID exists in the importing XSystem, the existing XSet shall be replaced by the imported XSet instance on commit of the imported XSet. An XSystem may generate a fatal or non-fatal error on commit of the imported XSet, if a field replacement would result in a violation of system field update rules (e.g., cause *.xset.retention.<retention id>.duration* to decrease), or if the imported XSet has a binding policy property whose policy cannot be enforced on the importing XSystem.

8.8.2 XSet Import Process: Policy handling

There are numerous problems with the description of import policy handling. On pp.91-92, replace all of the text starting after the 4 item enumeration under "Importing an XSet" and ending with the last sentence before Figure 49 with the following rewritten text:

When the XStream is closed, the XSystem shall validate the canonical XSet data. If the canonical XSet data is invalid, then the XStream.close shall cause a fatal error and the imported XSet instance shall enter a corrupted state. Note that an XSet instance in a corrupted state cannot be recovered, and all operations, except abandon, shall cause a fatal error.

The XSystem shall validate any exported policies to determine their applicability and effect on the importing XSystem. An XSystem policy specified as a policy property in the exported XSet is encoded as a policy element in the canonical format. An XSet property value that is represented in the exported XSystem policy and for which there exists a *XSet.get< actual value>* method (e.g. *XSet.getActualRetentionDuration*) is encoded as a property element within the policy element of the canonical format (See Section 8.8.4.1, " XSet Manifest XML Format"). The exported policy information shall be used by the importing XSystem to validate the exported policies.

Exported policies shall be validated or reconciled to the policies of the importing XSystem (before XStream.close returns control to the application) as follows:

- Nonbinding XSet policy properties:
  - — If the importing XSystem has a policy with the same name and updating the exported XSet's policy property values represented in the exported XSet's policy to the values provided by that XSystem policy does not violate any XSet system field update rule (e.g., causing *.xset.retention.<retention id>.duration* to decrease is a violation), then the exported XSet policy field is valid and shall be included in the imported XSet instance.
  - — If the importing XSystem has a policy with the same name and updating the exported XSet's policy property values represented in the exported XSet's policy violates an XSet system field update rule or if the importing XSystem has no policy with the same name, then the importing XSystem may validate the exported XSet policy by changing the XSet policy property in the imported XSet instance to a policy name that does not result in a violation of the XSet system field update rules.
  - — Otherwise, the exported XSet policy field is invalid for the importing XSystem; the XStream.close for the import XStream shall cause a fatal error and the imported XSet instance shall enter a corrupted state.
- Binding XSet policy properties:
  - — If the importing XSystem has a policy with the same name and updating the exported XSet's policy property values represented in the exported XSet's policy does not violate any XSet system field update rule (e.g., causing *.xset.retention.<retention id>.duration* to decrease is a violation), then the exported XSet policy field is valid and shall be included in the imported XSet instance.
  - — If the importing XSystem has a policy with the same name and updating the exported XSet's policy property values represented in the exported XSet's policy violates an XSet system field update rule, then the exported XSet policy field is invalid for the importing XSystem; the XStream.close for the import XStream shall cause a fatal error and the imported XSet instance shall enter a corrupted state.
  - — If the importing XSystem has no policy with the same name, then the importing XSystem may validate the exported XSet policy by creating a policy with the same name with policy property values that are not in violation of any XSet system field update rule and may include the XSet policy property in the imported XSet instance.
  - — Otherwise, the exported XSet policy field is invalid for the importing XSystem; the XStream.close for the import XStream shall cause a fatal error and the imported XSet instance shall enter a corrupted state

Upon successful XStream close, the imported XSet instance contains validated fields including any XSet policy properties . The imported XSet instance shall enter the dirty state (meaning it has been modified), and *.xset.xuid* shall be populated with the XUID from the imported data. The XSet instance will not have been committed; it may then be committed or discarded as any normal XSet instance would be. Unless there is a subsequent change to a binding field in the XSet, the XUID in *.xset.xuid* will be returned on successful commit of the imported XSet instance.

If the XSet already exists in the XSystem, as determined by *XSystem.accessXSet*, the application can open the existing XSet and compare the two so that any conflicts can be resolved, if needed. Comparing the two allows the application to do a more sophisticated import, where it can choose to commit (or not commit) the XSet without any consequences.

On successful import, the system fields shall be modified as shown in Table 49.

8.8.4.1 XSet Manifest XML Format: Clarify policy

The fourth sentence in the second paragraph of this section ends with:

> followed by a value element containing the value of the property.

Insert the word "policy" before "property" for clarity.

The sixth sentence in that paragraph is:

> The property elements contained in the policy element shall correspond to the policy subtypes.

That's incorrect, as there's no such thing as a policy subtype. Replace that sentence with:

> The property elements contained in the policy element shall correspond to the actual values of the XSet properties represented by the XSystem policy.

Insert the following text after the second paragraph:

> The following is an example of a policy element:

```
<policies>
 <policy name=".xsystem.management.policy.list.default"
type="application/vnd.snia.xam.string" readOnly="false" binding="false"
length="7" >
    <string>default</string>
    <property name=".xset.retention.base.duration"
type="application/vnd.snia.xam.int" readOnly="false" binding="false"
length="8"  >
       <integer>60000</integer>
   </property>
   <property name=".xset.retention.base.enabled"
type="application/vnd.snia.xam.boolean" readOnly="false" binding="false"
length="1"  >
       <boolean>true</boolean>
   </property>
    <property name=".xset.retention.event.duration"
type="application/vnd.snia.xam.int" readOnly="false" binding="false"
length="8"  >
       <integer>0</integer>
   </property>
```

```
   <property name=".xset.retention.event.enabled"
type="application/vnd.snia.xam.boolean" readOnly="false" binding="false"
length="1"  >
       <boolean>false</boolean>
   </property>
    <property name=".xset.deletion.autodelete"
type="application/vnd.snia.xam.boolean" readOnly="false" binding="false"
length="1"  >
       <boolean>true</boolean>
   </property>
   <property name=".xset.deletion.shred"
type="application/vnd.snia.xam.boolean" readOnly="false" binding="false"
length="1"  >
       <boolean>true</boolean>
   </property>
  </policy>
</policies>
```

8.8.4.2 The Canonical Representation: Clarify what offset points to

The offset in a table of contents line is described as:

- [offset in bytes] is the offset in bytes from the beginning of the package to the start of the binary data in the package

Change "to the start of the binary data" to "to the start of the MIME boundary delimiter that precedes the Content-Type description of the binary data".

8.8.4.3 XSet Export Example: Example Table

The value for the boolean example property is correct but unclear. In Table 51 change the Value entry for the *org.snia.property.test* row from "1" to "true" because that is a boolean property.

8.8.4.3 XSet Export Example: MIME type problem

The MIME type for the XStream in the example is incorrect. In the following line of XML on p.98:

```
<xstream binding=”false” type=”mime/jpeg” length=”20000” readOnly=”true”
        name=”org.snia.stream.property”>
```

Change "mime/jpeg" to "image/jpeg".

8.9 XAM Jobs and XAM Job Control: Field Names

At the end of this section, replace the incorrect *.xsystem.job.query.continuance.supported* field name with the correct *.xsystem.job.xam.job.query.continuance.supported* field name.

8.9.2 Standardized Job Output Fields: errorhealth field

The last two sentences in this section currently read:

> The health field describes if the job is okay or if it has encountered an error. The error field will only be created, if the job has encountered an error, and will contain the specific error token associated with the job.

Both fields are only created when an error occurs. Change the above sentences to:

> The health field indicates whether the job has encountered an error. Both the health and error fields will only be created if the job has encountered an error. The error field will contain the specific error token associated with the job.

8.9.2.1 Job Status: field name

Change the name of the job status field from *.xset.job.status* to the correct *.xam.job.status* .

8.9.2.2 Job Error: field names and errorhealth field value

Change the name of the job errorhealth field from *.xset.job.errorhealth* to the correct *.xam.job.errorhealth* .

This errorhealth field is only created when a job error occurs and hence can only have an error value. Remove the words ' either be “OK” or ' from this sentence in the paragraph following the name of the field:

> The value shall either be “OK” or “ERROR”.

Change the name of the job error field from *.xset.job.error* to the correct *.xam.job.error* .

8.10.1 The XAsync Object: halt clarifications

In the last paragraph on p.103 (paragraph begins with "Once an asynchronous operation has begun,"), after this sentence:

> XAsync.halt is a blocking method that will not return until the asynchronous operation has been successfully halted or completed.

Add the following sentence:

> If the asynchronous operation has been successfully halted or completed when XAsync.halt is invoked, the method shall return immediately.

8.10.2 XAsync FSM: Table 54 state name

The third entry in the Start State column is Abandon, which is not a state in the XAsync FSM. Change this entry to Complete to match Figure 20 and the rest of the Table.

8.10.2 XAsync FSM: halt clarifications

In Figure 20, add XAsync.halt to the list of operations for the transition arrow from the Complete state to the Complete state.

In Table 54, for the Complete state (renamed from Abandon by an erratum above), add a new row after the XAsync.close row, with the following contents:

- Transition: XAsync.getStatus XAsync.getXSet XAsync.getXUID XAsync.getBytesRead XAsync.getBytesWritten XAsync.halt
- Final State: Complete
- Comment: The XAsync FSM shall stay in the Complete state. XAsync.halt shall return immediately.

9.2.1 XSet Retention: allow zero duration

In the paragraph describing *.xset.retention.<retention id>.duration*, change the following sentence:

> This value shall be specified as a positive number of milliseconds, or as “forever,” which is represented by a value of -1.

to:

> This value shall be specified as a positive number of milliseconds, zero, or as “forever,” which is represented by a value of -1.

in order to allow a zero value for duration.

9.2.1.1 XSet Retention: event retention is always binding

In the paragraph describing the XSet.createRetention method, the last sentence is:

> An application shall also specify a xam_boolean to select, if .xset.retention.list.<retention id> shall be binding or nonbinding.

After that sentence, add the following sentence:

> If the retention id is "event", and this xam_boolean value is FALSE, then a non-fatal error shall be returned because the *.xset.retention.list.event* property is always a binding property.

in order to avoid creation of non-binding "event" retention..

9.2.1.2 XSet Retention Management FSM: allow zero duration

In Table 59 - Setting the Duration, in the Comment boxes for the XSet.setRetentionDuration (initial) transitions from both the Indefinite (enabled) and Inactive states, change "Initial duration value can be any valid positive integer or -1," to "Initial duration value can be any valid positive integer, zero or -1," in order to allow a zero value for duration.

9.2.2 XSet Deletion: Field Names

In Table 62 - Deletion Value Management Properties:

- In the Comments box for the *.xset.deletion.autodelete* field, replace the incorrect *.xsystem.autodelete* field name with the correct *.xsystem.deletion.autodelete* field name.
- In the Comments box for the *.xset.deletion.shred* field, replace the incorrect *.xsystem.shred* field name with the correct *.xsystem.deletion.shred* field name.

In the first line of the third paragraph describing *.xset.deletion.shred* change the incorrect *.xset.shred* field name to the correct *.xset.deletion.shred* field name.

9.3.3 XSet Management Policy: <name> usage

In the last paragraph of the descriptions of:

- *.xset.retention.<retention id>.enabled.policy*
- *.xset.retention.base.duration.policy*
- *.xset.retention.<retention id>.duration.policy*

Change the <*policy name*> token in the field name at the end of the sentence to <*name*>.

10.3 Introduction to the Query Language Grammar: Use a valid field name in example

In the first query example, change ".xam.time.xuid" to ".xset.time.xuid".

10.4 Level 1 Query: Where Clause Operators

Add boolean comparisons to Table 65. Specifically:

- Insert a new row above the "int" row with "boolean" in its left-most cell,
- Insert a new column under the Field Type column spanner to the left of the xam_int column,
- Put "xam_boolean" in the topmost (Field Type) cell of the new column and a "*" in the cell immediately below it (allow a boolean literal to be compared to a xam_boolean). Leave all other cells in the new column blank (booleans cannot be compared to non-boolean types).

10.4.4 Field Attribute Accessor Functions: Typo

The first example for readonly() contains an extraneous angled double close quote near the end of the line - remove it.

10.4.5 Logical Operators: Error in footnote

In footnote 1 on p.136, change the double equals ("= =") to a single equals ("=").

10.4.6 New Section: Comparison to non-existant fields

After Section 10.4.5, add a new Section 10.4.6 with a section title of "Comparison to non-existant fields" and the following content:

> If a field referenced in a XAM Query string does not exist in an XSet, then:

- If the field name is the argument to the exists() function, then the exists() function shall return FALSE for evaluation of the query on that XSet.
- Any other use of the field name in the query string shall cause an XSet in which the named field does not exist to be excluded from the result set.

10.5 Level 2 Query: Where Clause Content Search Operators

In the first paragraph of this section, replace the incorrect *.xsystem.job.query.level2.supported* field name with the correct *.xsystem.job.xam.job.query.level2.supported* field name.

The current text that specifies the "within" operator is:

- within – Indicates that the specified stream contains the indicated tokens near each other. This operator shall accept three arguments, two string tokens, and the token distance allowed. There shall be no implied ordering for the two token. For example,

  select ".xset.xuid" where "com.example.mystream" within('word1', 'word2', 7)

  This example indicates that the word 'word1' occurs within 7 tokens of 'word2.'

Add the following text immediately after the above text:

Distance between tokens can be explained by the following example:

```
A quick brown fox jumped over the lazy dog
```

The distance between tokens shall be defined as the number of intervening, non-matching tokens between the two tokens specified to the *within* operator. For the previous example, the distance between *dog* and some other tokens are listed here:

- lazy – Zero. There are no tokens between *lazy* and *dog.*
- the – One. There is one token between *the* and *dog.*
- over – Two. There are two tokens between *over* and *dog.*

The distance between any token and itself shall be considered to be zero, and shall be equivalent to a *contains(token)* clause in the query.

Negative distance values are not meaningful and shall generate a query parse error.

10.6 Complete Grammar: Binary Operation is Required

The ABNF rule for binary-op is:

```
binary-op = "=" / "<>" / / "!=" / "<" / "<=" / ">" / ">=" / "like"
```

The occurrence of doubled slashes (/ /) allows the binary operator to be omitted. Change this to a single slash (/) .

10.6 Complete Grammar: Allow property name as factor.

In the section for factor, add a line for a property name to allow for boolean property expressions. The line to be added is:

```
            (property-name)  /
```

The resulting rule should be:

```
factor = ( property-name binary-op literal-expression ) /
         ( property-name ) /
         ( attribute-bool-fn ) /
         ( attribute-bool-fn binary-op literal-expression ) /
         ( attribute-function binary-op literal-expression ) /
         ( property-name level2op )
```

10.6 Complete Grammar: Literals

The portion of the ABNF grammar that defines literals for the XAM query language is currently:

```
literal-expression    = ( ["+" / "-"] (xam-integer-literal / xam-double-literal) /
                        xam-string-literal / xam-boolean-literal /
                        xam-date-literal/
                        xam-xuid-literal )
xam-boolean-literal   = "TRUE" / "FALSE"
xam-integer-literal   = 1*digit
xam-double-literal    = ( *digit) "." (*digit) [exponent] /
                        (*digit) [exponent] /
                        "NaN" / ["+" / "-"] "Inf"
exponent              = ("e" / "E") ["+" / "-"] 1*digit ; base 10 exponentiation
xam-string-literal    = sq *(char) sq       ; String as in 'abc def ghi'
xam-date-literal      = "date" "(" xam-string-literal ")"; date( '2006-04-01T00:00' )
xam-xuid-literal      = "xuid" "(" xam-string-literal ")"; xuid ( 'AAAAAwAbB68AAH' )
sq                    = %x27         ; This is a single quote, as in 'foo'
dq                    = %x22         ; This is a double quote, as in "bar"
```

The rules for integer literals and double literals have a number of problems that allow:

- Type ambiguity (e.g., the string "1" could be an integer or a double literal).
- Badly formed literals caused by missing digits (e.g., "." is an allowed double).
- Two sign prefixes on "Inf" (e.g., "+-Inf" is allowed).
- A sign prefix on "NaN", which is unsigned.

The corrections change the grammar to enforce the following:

1. A double constant requires a decimal point with a number before it and a number after it.
2. Any numeric component of a literal requires at least one digit
3. Two sign prefixes cannot be adjacent.
4. NaN is always unsigned.

The editing instructions are:

A. Remove the sign prefix and inner set of parentheses from the rule for `literal-expression` so that the rule becomes:

```
literal-expression = ( xam-integer-literal / xam-double-literal /
                       xam-string-literal / xam-boolean-literal /
                       xam-date-literal / xam-xuid-literal )
```

B. Replace the rules for `xam-integer-literal`, `xam-double-literal` and `exponent` with the following three rules:

```
xam-integer-literal = [sign-prefix] 1*digit
xam-double-literal  = [sign-prefix] (1*digit) "." (1*digit) [exponent] /
                        [sign-prefix] "Inf" / "NaN"
exponent      =    ("e" / "E") [sign-prefix] 1*digit
```

C. Add the following new rule after the rule for exponent:

```
sign-prefix   =    ("+" / "-")
```

10.6 Complete Grammar: Quotes

The angled characters used for ASCII single quote in the ABNF rules is not appropriate; replace them with characters that render the quotes as vertical rather than slanted.

10.6 Complete Grammar: Whitespace character rules

In the first paragraph in Section 10.6, change this sentence:

> The XAM QL is case insensitive and uses the US-ASCII [RFC 1345] character set.

to:

> The XAM QL is case insensitive and uses the US-ASCII [RFC 1345] character set, except that the control characters in the range 0x00 to 0x1F and the delete character (DEL, 0x7f) are not used.

After the ABNF grammar, add the following paragraph.

> Use of the space character (SP 0x20) has been omitted from the above grammar for clarity. When the space character occurs within a quoted string, it represents a space character. Within a XAM query expression the space character may be used before or after any key word, quoted string, binary operator, attribute function name, literal or any of the characters "(", ")" and ",". In some cases the space character is necessary, e.g., to distinguish "and not" (two key words) from "andnot" (not a key word). The space character shall not be used within any integer or double literal.

10.6.1 Reserved Key Words and Operator Precedence: Missing key words

Add "date" and "xuid" (without the quotes) to the list of reserved key words.

10.7.1 Query Job Specific XSet Fields: Readonly status

In Table 71, all of the entries in the Readonly column should be "TRUE when running". Change the latter three entries from "TRUE" to "TRUE when running".

10.7.2 Runtime Behavior of the Query Job: Field Name

In the first and second lines of the first paragraph of this section, replace *org.snia.xam.job.query.command* with the correct *xam.job.query.command*

10.7.4 Result Stream Format: End of Stream

At the end of the section, add the following paragraph to explain that the end of the result stream cannot be indicated while the query job that generates the results is running:

> The query result stream is logically defined to contain all of the query results, including those that have yet to be generated by a running query job. A XAM application that reads the entire results stream until end of stream is indicated does not need to recheck the results stream later to determine whether more results have been added. It shall be possible for a XAM application to read the XStream that contains the query results while the corresponding query job is running, but end of stream shall not be indicated before the job completes. If all of the available XUIDs from a query results XStream for a running query job have been read, and a significant delay is likely until another XUID is added, an XStream.read method invocation should return quickly, indicating that zero bytes have been read and the method shall not indicate that the end of the XStream has been reached. An asynchronous XStream.read invocation should wait for additional results while the corresponding query job is running.

10.7.6 Runtime Caveats: Halting a Job is an not an error.

The second sentence of last bullet of this section currently reads:

> When a query job is halted, the appropriate value shall be set on the standard job health and status fields (*.xam.job.errorhealth* and *.xam.job.status*, respectively).

Change it to the following two sentences:

> When a query job is halted, the appropriate value shall be set on the standard job status field (*.xam.job.status*). Halting a query job shall not cause creation of the standard job health and error fields (*.xam.job.errorhealth* and *.xam.job.error*, respectively).

10.7.9 What Is / Is Not Included in a Query Result: Health field may not exist

The first sentence of this section currently reads:

> The completeness of a query job can be evaluated by evaluating the health and status fields of the job XSet.

The health field may not exist. Replace this sentence with these two sentences:

> The completeness of a query job can be evaluated by evaluating the status field of the job XSet, and the health field (if it exists) of the job XSet. The health field is only created if the job encounters an error.

10.7.10 Query and Permissions: Authorization granule names

The names of the authorization granules in the second paragraph of this section are incorrect. Change "The query job authorization granule" to "the job authorization granule" and change "the job-query-commit authorization granule" to "the job-commit authorization granule" to correct this.

10.8.9: Add example of query with mixed types

Insert this new section containing an example after section 10.8.8:

> ### 10.8.9 Query with mixed types
>
> This query demonstrates querying a named property with mixed types:
>
> ```
> select ".xset.xuid" where ("com.example.bar" >= 10) or
>         ("com.example.bar" like '%ing')
> ```
>
> Result Set
>
> - XSET1
> - XSET2
>
> XSET1 is included in the result set because the value of “com.example.bar”='string' matches the expression *like '%ing'.* XSET2 is included in the result set because the value of “com.example.bar”=42 matches the expression *>= 10.*

Annex B.2 XSD: double and XUID types

The XAM floating point stype is a double. The XSD for the canonical interchange format incorrectly uses float; change that type to the correct double type by changing:

```
<xs:element minOccurs="1" maxOccurs="1" name="float" type="xs:float"/>
```

to:

```
<xs:element minOccurs="1" maxOccurs="1" name="double" type="xs:double"/>
```

The XSD is also missing the XAM XUID type. Immediately after the above line, insert:

```
<xs:element minOccurs="1" maxOccurs="1" name="xuid" type="xs:string"/>
```